Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

People matter, results count.

MANAGEMENT OF CLOUD SOURCED APPLICATIONS

A MANAGEMENT FRAMEWORK

Master Thesis Information Management

Anne Slaa

TILBURG UNIVERSITY
Understanding Society

MANAGEMENT OF CLOUD SOURCED APPLICATIONS
Thesis Master Information Management

Author
J.C. (Anne) Slaa

Tilburg University
Warandelaan 2
5037 AB Tilburg

Supervisor
prof. dr. W.J.A.M. (Willem-Jan) van den Heuvel

Capgemini Nederland B.V.
Papendorpseweg 100
3528 BJ Utrecht

Supervisor
M. (Marcel) Lamb

Information Management
School of Economics and Management
Tilburg University

Tilburg, 30 November 2012

MANAGEMENT SUMMARY

Software-as-a-Service (SaaS) refers to a software application that is being offered over a networked medium, for example the internet. The software application represents the top layer of three layers entailed by cloud computing: infrastructure (IaaS), platform (PaaS) and software (SaaS). Cloud computing is mainly characterized by on-demand provisioning over a networked medium, dynamical scaling and pay-per-use. Cloud computing reduces the technological overhead as well as management tasks for the customer. Cloud services can be distinguished by their deployment method: public, private, hybrid or community. The characteristics of cloud computing make it easy to start or stop using a service and therefore it fits temporary projects well and lowers the barriers to try something new.

Software development has been going through different cycles that made development more abstract and therefore better understandable. The most abstract layer is visual programming or model-driven development. Traditionally software was being developed by engineers writing source code. But with visual programming, a model defining the software flow, cohesion, state transitions and windows is created and the source code is generated automatically from this model. Software development can be approached in two ways: by ongoing incremental iterations or as a sequence in which the progress flows steadily downwards. A popular incremental method is called “agile” and involves the end-user in the development process. Agile development and SaaS deployment support each other because changes can be quickly shown to the end-user.

Traditional methods like the ITIL and ASL frameworks for IT management are built on practices for traditional software that is not provided on-demand, over the internet, easily scalable and paid per consumption. Traditional software is developed by programmers and installed on-premise. This requires all resources to be purchased upfront and IT management functions to be deployed locally. When deploying a software solution as SaaS, management tasks shift from the customer to the vendor, new roles can be involved and some management tasks should be set up differently. The ITIL 2011 version adapts changes to fit the service lifecycle better than earlier versions. For ASL, a proposal is available that takes some SaaS aspects into consideration. Both traditional frameworks, however, represent a part of the management functions and can be integrated into one framework that covers all management functions. This is exactly what Capgemini has been doing for traditional IT management by creating their Way of Working framework.

The literature is combined to form the first concept of the theory, which consists of the SaaS management framework, the change management process and the configuration management process. The framework is the set of processes at the highest abstraction level, meaning that it describes which management processes may apply and how they interact. To find whether SaaS deployment and model-driven development involve changes at process-level, the change and configuration management processes have been studied. These processes are present in all versions of ITIL and ASL and likely to be still applicable in a SaaS deployment environment. Both these processes belong to the framework’s core and interact with each other.

The first concept of the developed theory was tested by conducting case studies. The case studies were performed to test the two subjects of this research in practice. SaaS deployment and model-driven development are both aspects supported by the Mendix development platform. The first case study was conducted at PKN, the organization that uses and maintains the largest Mendix-based application currently deployed and a pioneer in the use of the platform. The second case study was conducted at Nobel, an organization that provides hosting, development and management services. The first organization has implemented IT management on a single Mendix application and the latter manages multiple Mendix applications. After the theory testing phase, the theory has been modified.

In the final phase, the theory has been validated by IT management and development experts within the companies PostNL and Capgemini. The validation was approached similarly to the case studies, but left out the questions that are related to the theory indirectly. After a total of four case studies in two separate phases, the theoretical framework has been adapted to practice. Important findings during the case studies were that cloud computing is likely to involve multiple vendors due to its nature. The end-customer and every vendor have their own role in the management of the service and its underlying resources. This differs per application, but the distinction between different functions is shown in the framework. The management aspects “chain management”, “customer success management” and “access management” were discovered as new processes and other processes were merged or placed at another level in the service lifecycle.

PREFACE

With this master thesis I complete the Information Management Master program at Tilburg University. Slightly over two years ago I started the preliminary pre-master program at Tilburg University that after one year got me into the Master program. The pre-master followed up on the Bachelor of Information & Communication Technology I completed at Fontys University of Applied Sciences in Eindhoven.

The final phase of writing my master thesis took me three months longer than the Information Management program dictates. My focus was therefore not on finishing within the academic year, but on conducting a complete and valid research that is also usable in practice. The university and my university supervisor really supported this by providing good feedback while also having high expectations. This could be afforded by the small and intimate scale of the Information Management department at the university.

This thesis describes the research I did on the management of Software-as-a-Service (SaaS) applications. For traditional applications, frameworks like ITIL and ASL are nowadays considered to describe a de-facto standard set of processes for shaping effective, efficient and mature management processes for technical, functional and application management. However, these frameworks are based on good practices that may not be applicable in a cloud environment. Since model-driven development is also emerging with this trend of cloud deployment, SaaS and model-driven development are the studied subjects of this research.

The research subjects result from the initial conversation I had with Capgemini operational manager Marcel Lamb. This conversation clarified that many large companies, among which Capgemini, are currently seeking how they should use these emerging technologies. The companies do not question whether and how they should do the development, since that has already been going on for some years, but nowadays the interest is mainly in how these technologies need to be managed after they have been developed.

I would hereby like to thank everyone who helped me with my research or was somehow involved in it. I would especially like to thank my university supervisor for the ideas and insights resulting from the regular conversations we had. I thank all people who participated in the two case studies and the two validation cases involving expert reviews. Without the interviews you were willing to participate in, the theory developed in this research could not have been adapted to the use in practical situations. I would like to thank my parents for reading my thesis and providing their feedback, I thank the study association ASSET | SBIT for organizing some memorable activities and always being there for a break and I thank my family and friends for the support they gave me during my study and the months of writing my thesis. Above all, I thank God for the opportunities He brought on my path.

Anne Slaa
Tilburg, November 2012

TABLE OF CONTENTS

FIGURES AND TABLES

INDEX OF FIGURES

INDEX OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
ASL	Application Services Library
ASP	Application Service Provider
BiSL	Business information Services Library
BPaaS	Business Process-as-a-Service
CDMI	Cloud Data Management Interface
CI	Configuration Item
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CSD	Custom Software Development
DC	Distributed Computing
DSDM	Dynamic Systems Development Method
FDD	Feature-Driven Development
FS	Financial Services
ERP	Enterprise Resource Planning
HuaaS	Human-as-a-Service
IaaS	Infrastructure-as-a-Service
IS	Information Systems
ITIL	Information Technology Infrastructure Library
ITIM	IT Infrastructure Management
ITSM	IT Service Management
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
NIST	National Institute of Standards and Technology
OCCI	Open Cloud Computing Interface
OS	Operating System
PaaS	Platform-as-a-service
PC	Personal Computer
RAD	Rapid Application Development
SaaS	Software-as-a-Service
SLA	Software License Agreement
SOA	Service Oriented Architecture
UC	Ubiquitous Computing
WoW	Way of Working
XP	Extreme Programming

1. INTRODUCTION

Functional, technical and application management is described by different frameworks, from focusing on high-level strategic aspects of IT Governance to operational aspects of IT Service Management. Management, control, governance and maintenance are all terms that are used frequently in the environment where software is being used and IT and business need to be aligned. Now the development of software is shifting from traditional to online services in the cloud, Capgemini is one organization that likes to develop knowledge on how to manage the cloud services they provide and what the impact will be on their pricing models, since the largest portion of revenues is typically gained in the functional management phase, not in the development phase of applications.

1.1. RESEARCH SETTING

The research will be performed under commission of Capgemini Nederland B.V., the Dutch branch of a large worldwide consultancy firm. The advantages of this setting are on the one hand the ability to make use of practical knowledge that has been collected by years of experience and on the other hand the connection to Capgemini application management employees and customers.

1.1.1. PROFILE CAPGEMINI

Capgemini was founded in 1967 by Serge Kampf in France as a management consulting, outsourcing and professional services firm. Capgemini is operating in the technology sector, in the industry of Information Technology Services. Nowadays, the firm employs over 8,500 people in the Netherlands, 120,000 worldwide and is active in 40 countries. A noticeable growth area is India, which has grown fast in the last five years to approximately 40,000 employees today. Capgemini's six strategic partners are: EMC^2, HP, IBM, Microsoft, Oracle and SAP. The global revenues and net result have shown a strong upward trend in the past three years, the net result even rose with 44.3% between 2010 and 2011 to 404 million euros (Capgemini, 2011).

Cap Gemini S.A. is the name of the company's share on the stock exchange. Cap Gemini S.A. is one of the world's largest providers of systems integration and consulting services. Two companies are active under this firm: Capgemini and Sogeti. Capgemini provides a range of business process outsourcing services in the areas of customer relationship management, finance, human resources and supply chain management. To deliver these services the company operates in the following disciplines: Capgemini Consulting, Application Services, Infrastructure Services, Business Process Services and Sogeti. The Sogeti division provides support and consulting to smaller customers through local offices. These disciplines are individually treated with their specific economic rules, but are also managed based on their own profits. Ultimately, the umbrella organization is Capgemini Group HQ in Paris, which decides on the goals per discipline and division.

The worldwide top competitors of Capgemini are: Accenture plc, HP Enterprise Group and IBM Global services. Figure 1 shows the 14 main competitors of Capgemini.

To provide business process outsourcing, Capgemini uses, amongst others, the software capabilities of two major ERP software brands: SAP and Oracle.

Figure 1: Worldwide main competitors of Capgemini

1.1.2. STRUCTURE CAPGEMINI

The organizational structure adopted by Capgemini is a functional structure. This should reduce the risk and vulnerability of the company to market changes and competition by expanding its business operations. For example; each discipline has its own chief executive. These sub-corporations are each subdivided on functional basis (e.g. Sales, HRM, Finance, Production, etc.) enabling both specialisation and economies of scale.

Capgemini is constantly expanding, both by growth of the current operations and divisions and external expansion. External expansion consists of three options:

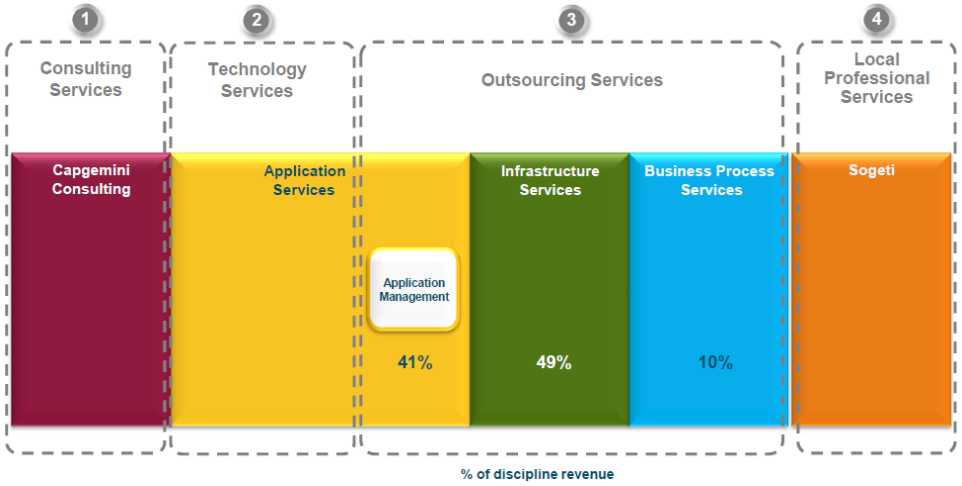


Figure 2: Capgemini disciplines & working areas

- Horizontal Expansion – ensured by new mergers & acquisitions
- Vertical Expansion – taking over companies in the supply chain
- Diversified growth – the mergers/acquisitions in new business segments/markets

Capgemini has a complicated organizational structure. Figure 2 shows how the disciplines and general working areas (Consulting Services, Technology Services and Outsourcing Services) combine to explain how the structure is translated into practice. The various markets would in this view be spread on a horizontal level over the different disciplines. The business units Application Services NL 1 (Apps 1) and Application Services NL 2 (Apps 2) are a combination of Technology Services and part of Outsourcing Services. Apps 1 in the Netherlands focuses only on Financial Service (FS) companies and Apps 2 handles all remaining industries. Apps 2 (figure 3) has therefore been divided into a public and private sector division, which both have a very similar internal structure. Both divisions have a practice Custom Software Development (CSD) which is concerned with the development of software solutions. The lifecycle of applications starts with the analysis of business requirements and customer advisory. Then, the design, development and verification of software solutions takes place and when the solutions have been developed, the management, maintenance and governance of the information systems needs to be covered. All the activities in this process are provided by Capgemini to their customers.

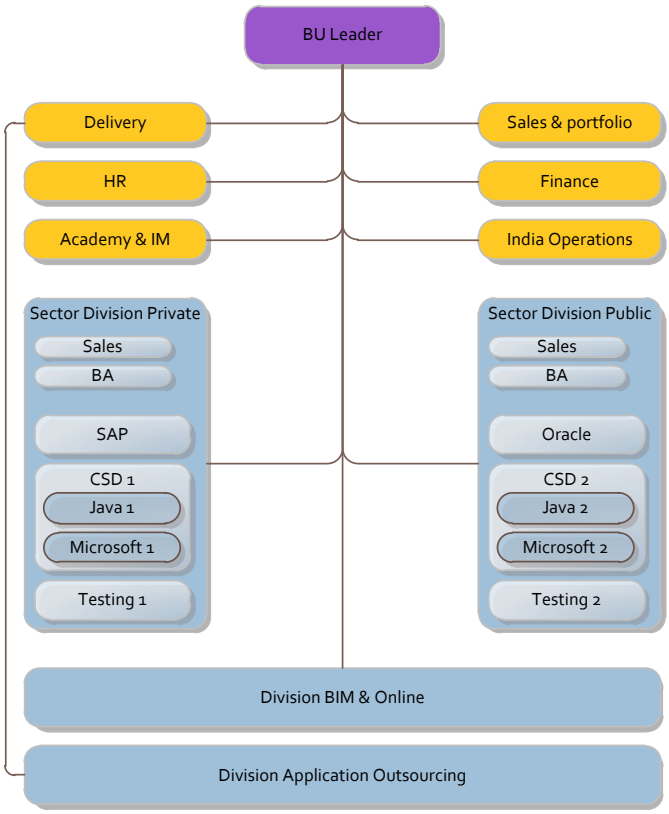


Figure 3: Structure Capgemini business unit "Application Services NL 2" (Apps 2) with division sectors

1.1. RESEARCH MOTIVATION

In the history of the design and maintenance of applications, management of the complexity has always been an issue. Gradually, solutions emerged, first with structured design (modularity), next with object-orientation and finally with the transition into Service-Oriented Architecture (SOA). Every step in this evolution concealed more of the application, thus organizing its complexity (van der Pols, 2008). Service-oriented computing is nowadays gaining more and more acceptance from organizations and becomes a popular method to deliver functionality against lower costs than traditional software architectures require for the development and maintenance of native systems. The idea of coupling service processes to a data structure makes the users more independent and makes the organization more flexible (Papazoglou & van den Heuvel, 2007).

Hassanzadeh, Namdarian, & Elahi (2011) state that the trend of loosely coupled systems is being adopted by organizations to *"maximize interoperability, reusability, flexibility and cost efficiency"*. However, new solutions raise new problems and flexible solutions need to be connected and integrated into each other. Lheureux, Thompson, Skybakmoen, & Wilson (2010) state that integration problems are growing as quickly as new solutions are being adopted. Bittman (2011) argues that virtualization should not be handled by a single project, but should be considered the beginning of a roadmap towards cloud computing. If there is strategic planning, problems like bad integration with other services and the organizational environment will rise in an early stage and can then be planned. Little (2008) analyzes a letter Gartner analysts posted on the Gartner Blog Network. The letter is intended amusing, but mentions the thorny issues of some SOA efforts. The letter for example mentions that SOA initiatives were not in correspondence with the business needs, reuse was not considered and there was no collaboration between teams. A survey of R. Smith (2009) confirms that SOA was not widely adopted and companies that did use SOA had not made systems available for external use. However, one could say that SOA already 'evolved' to the next big thing: cloud computing. SOA will not have the focus in this thesis since the technology of cloud computing has become very mature in itself. This thesis will study the cloud computing lifecycle at the Software as a Service (SaaS) level.

Cloud computing and Cloud sourcing are generic terms that may refer to any form of outsourcing the hosting and physical resources for applications, data and storage to another party that provides the services over the web. This can be broken down into three levels of services: Cloud SaaS, Cloud Platform as a Service (PaaS) and Cloud Infrastructure as a Service (IaaS). These levels refer respectively to functionality in the form of a software package or module, to the software platform for applications/systems to run on and to the infrastructure and hardware that is being used to host, access, store and process the former services (Vaquero, Rodero-Merino, Caceres, & Lindner, 2009). When comparing these terms to a native application on a personal computer, the SaaS layer would refer to the user application, the PaaS layer to the operating system (OS) including development tools and the IaaS layer to the hardware and network infrastructure.

Traditionally, businesses had the choice to keep the software hosted in-house, with the need to make the capital expenditures to have all necessary resources (e.g. hardware, infrastructure, software, IT personnel/skills and datacenters) to be able to provide full software functionality at all times. Later, kinds of outsourcing were invented by which other companies would physically host the software and provide it on a contractual basis, shifting the expenditures from capital to operational and giving organizations the chance to focus more on their core business. However, these contracts tended to be sealed for long periods of time and provided the user with only limited flexibility. Nowadays new companies provide these software services "in the cloud", which is really on demand delivery of resources and functionality, making the use of resources and thus the costs more flexible. Also, a shift in focus took place by software application manufacturers (e.g. ERP manufacturers) and companies that use the software from aiming to provide all business functionalities in one large system to modularly built software that is easy to couple and decouple to other (existing) systems or to the existing infrastructure. This approach gives users the advantage of being able to select the best module for every single specific functionality, since they are no longer bound to one software package or vendor. The vendor's advantage is that they can expertize and excel in areas which they want to provide as their core functionalities or modules, instead of having to provide a bit of everything for every industry.

Another shift in paradigm is model-driven architecture becoming more mature. Model-driven software development can provide customer software without being custom coded by programmers. Certain PaaS providers integrate development tools that support model-driven engineering in their system. This form of software development allows users to easily create a program from a design model without the interference of a programmer (France & Rumpe, 2007). These platforms make it a lot easier to create a new software module or system and this raises questions on the approach of the next process step in the software lifecycle where things like management, maintenance, governance and control come in. When users or business analysts can create their own software or module without the need of having in-depth knowledge on development constraints and data-structures, do traditional methods then still provide sufficient directives for the management and maintenance of applications? Accepted traditional methods for management are frameworks that were based on best-practices for traditional software, e.g. the Information Technology Infrastructure Library (ITIL) for technical and infrastructure management, Application Services Library (ASL) and ITIL for application management and Business Information Services Library (BiSL) for functional management (van Wamelen, 2010). The division into these three management positions was traditionally proposed by Delen & van Looijen (1992) and further refined in 2004 (van Looijen, 2004). The ITIL framework covers all IT Service

Management (ITSM) stages, which is the process for effective and efficient management and control of the quality of IT services.

In the trend of cloud computing, certain companies have specialized in providing software functionality in the cloud that organizations formerly only used as native PC or server application (e.g. ERP, CRM, Office). An example of such a company is the international organization Salesforce.com from California (Salesforce.com, 2012). This organization provides business software for Customer Relationship Management (CRM) as a cloud service. Another company providing cloud services is Mendix, originally a spin-off from the Technical University Delft and the Erasmus University of Rotterdam, based in Rotterdam and denominated "Cool Vendor 2009" by Gartner. This Dutch organization has developed a platform tool that makes it easy to plan, develop, deploy and manage custom applications that can be integrated with existing systems (Mendix, 2012). Also Be Informed is a Dutch organization providing a cloud platform for model-driven development and management (Be Informed, 2012). The software from the two latter mentioned companies can be classified as PaaS and the first as SaaS, however, Salesforce.com also provides a PaaS, called Force.com and an IaaS/PaaS combination which is called "Heroku".

1.2. RESEARCH OBJECTIVE

The goal of this research is to develop the generic set of processes (a framework) that are necessary to effectively manage applications that run as SaaS services and are developed in a model-driven way. The purpose of this framework is to provide generic directives that can be used to shape the management framework for a specific application that has been developed in a model-driven way and is deployed in a SaaS environment.

Existing frameworks for IT management are often based on best-practices that originate from traditional software and may therefore not fit cloud services on aspects like contract management, availability management and change management. These existing frameworks, however, provide a basis to come to the SaaS framework. When combining them with the distinguishing characteristics of cloud services compared to traditional deployment and model-driven (and agile) engineering compared to traditional programming. The projected result of this research is a set of management processes which together form a framework that fits the management needs for SaaS services.

This study may conclude that certain aspects of traditional IT management are inefficient and should not be implemented or implemented as an adjusted process. Another result might be a decision tree or decision matrix which makes managers easily decide and argue the best shape for the IT management process for a specific type of software service. This study could also result in a process description, based on traditional (e.g. ITIL) methods but revised in order to work effectively and efficiently with cloud software. Alternatively, the research results may show that traditional methods are actually sufficient in the cloud context or that the traditional methods should be extended with new best-practice insights specific for cloud software.

1.3. PROBLEM STATEMENT & RESEARCH QUESTIONS

When an organization has made the decision to source certain functionality to a cloud provider and developers have built this software service, the involved organizations will need the management, maintenance, governance, and control of these applications to be organized, because e.g. changes might be required or incidents and problems might occur when users start working with the application. Traditionally, companies would make use of frameworks based on best-practices to organize the management, e.g. ITIL and ASL for technical and application management. Capgemini is a consultancy and implementation partner for many top-200 organizations in the Netherlands. When they have implemented a cloud solution for their customers, both Capgemini and the client company need to make a decision on how this solution should be managed and who will be responsible for which management process.

RESEARCH QUESTION

How should the set of management and control processes in general be made up for SaaS applications that were developed using model-driven technologies?

SUBQUESTIONS

1. What are the distinguishing characteristics of cloud computing compared to traditional software?
2. How do model-driven and agile development differ from traditional development methods?
3. What are effective methods for operational management of traditional IT applications?
4. How should traditional frameworks for IT management and maintenance be adapted to provide effective directions in a model-driven SaaS environment?
5. How can the adapted management framework be developed and validated for practical use?
6. In what way are the change and configuration management processes affected?

1.4. RESEARCH METHOD

This research has a theoretical and an empirical part. The first part is being *exploratory* to find the characteristics of cloud computing, to describe existing frameworks for management, and to develop a conceptual SaaS management framework. The exploration is there to define the subjects and to explain from theory how the management framework should differ from management and maintenance of traditional software. The second part is being *constructive* in developing, testing and validating a framework that complies to SaaS management, maintenance and control needs. This part focuses on the practical alignment of the framework and processes, to maximize their efficiency and effectiveness.

The research method will be qualitative for two reasons. First, a framework that is suitable for practical needs can only be designed considering different real-life cases. The cases will contribute to the construction in an incremental manner. Second, no cases combining the subjects of operational management and SaaS services can be found in current scientific literature nor in other literature to validate a quantitative research. The required knowledge can be obtained from the IT management experience of companies that currently deploy SaaS applications. Extraction and combination of these pieces of information will lead to a general management framework that provides structure and assurance to companies still hesitating to start using cloud technology services and that can be used for shaping the management processes of a specific application.

The used research approach relies on both theory and practice to extract relevant information about the management processes for cloud software. A risk may be that managers and users at the studies companies are insufficiently able to provide this information since the subject is too new to them. This risk should be avoided by interviewing the people that are involved in the management of a model-driven developed SaaS application. A plus would be if the organizations managing a SaaS application are also familiar with traditional IT management for a similar application.

The design science research method is used as explained by Hevner et al. (2004). The function and rationale of design science will be explained in the following paragraph. The research design paragraph describes the research model and method used.

1.4.1. DESIGN SCIENCE RESEARCH

In general, research can be defined as the activity that contributes to the understanding of a phenomenon. The phenomenon is typically a set of behaviors or facts belonging to an entity or entities. A naturally occurring phenomenon may be explained or predicted or the phenomenon may be created in extension to the boundaries of existing phenomena. The latter is what design science research is about. Design deals by definition with the creation of something that does not yet exist (Kuhn, 1970; Vaishnavi & Kuechler, 2004). There is a clear distinction in definition between natural science, e.g. the *behavioral science* paradigm, and the *design science* paradigm, which Simon (1996) refers to as "*science of the artificial*":

"*A natural science is a body of knowledge about some class of things – objects or phenomena – in the world: about the characteristics and properties that they have; about how they behave and interact with each other. […] A science of the artificial is a body of knowledge about man-made objects and phenomena designed to meet certain desired goals*" (Simon, 1996).

Design science research in Information Systems (IS) is a different research perspective and a relatively new approach to conduct IS research, next to the positivist and interpretive approaches of natural science. It can be defined as follows:

"*Design science research involves the design of novel or innovative artifacts and the analysis of the use and/or performance of such artifacts to improve and understand the behavior of aspects of Information Systems. Such artifacts include – but certainly are not limited to – algorithms (e.g. for information retrieval), human/computer interfaces and system design methodologies or languages*" (Vaishnavi & Kuechler, 2004).

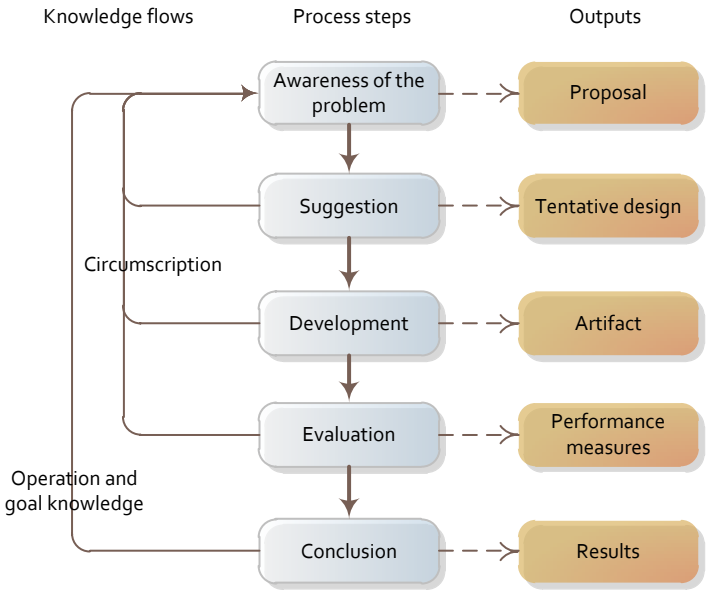


Figure 4: general methodology of design research (Vaishnavi & Kuechler, 2004)

The methodology of design science research is shown by the model in figure 4. Vaishnavi & Kuechler (2004) state that the first part of the process involves abduction; the data description in the proposal and tentative design lead to a hypothetical artifact or artifacts in the development stage. Then, from the development stage there is a deductive logic by reasoning from the artifact's properties to a logical conclusion.

Hevner et al. (2004) propose seven guidelines that assist researchers to fulfill the requirements of effective design science research, however these are not mandatory or equally applicable to every study. The first guideline states that the result of design science research must be a feasible artifact (which can be a construct, model, method or instantiation) that is described effectively. The second guideline describes that the objective is to develop technology-based solutions to unsolved relevant business problems and the third guideline states that the efficiency and effectiveness of the design artifact must be well demonstrated through well-executed evaluation methods. The fourth guideline prescribes that the research must provide clear contributions to the research areas of the design artifact, design construction knowledge and/or design evaluation knowledge. The fifth guideline addresses the way research is conducted. The research must be constructed with respect to the applicability and generalizability of the artifact. The sixth guideline states that design should be iterative to come to the best or optimal solution that is feasible in the business environment. The nature of the design process is described by Simon (1996) as a generate/test cycle. The seventh guideline emphasizes that the research is well-communicated to both technology-oriented and management-oriented audiences. This means that the presentation of the research contains enough detail on how the artifact is constructed and implemented and enough detail to determine whether management needs to free resources for constructing or purchasing and using the artifact in their organization.

To conclude, the behavioral science paradigm is effective in search of "what is true" and the aim of the design science paradigm is to create "what is effective" (Hevner et al., 2004). This thesis is not written to explain a current framework but to create an effective framework and process models for a new environment that does not yet have a (suitable) framework.

1.4.2. RESEARCH DESIGN

The interest of this research is in two characteristics of applications: SaaS deployment on a platform (PaaS) and being developed in a model-driven fashion. The research sponsor Capgemini works with multiple PaaS vendors which provide integrated development tools. These tools may support three different types of service development: code-driven (e.g. Force.com), model-driven (e.g. Mendix and Be Informed) and template-driven (e.g. Windows Azure). Two PaaS vendor companies have been found in the literature that would be eligible for this research since their platforms fully support both characteristics: Mendix (Henkel & Stirna, 2010) and Rollbase (Lawton, 2008). The Salesforce Heroku, Microsoft Azure and open source VMWare Cloud Foundry platforms do not fully support model-driven development and Be Informed is not available as a public cloud solution. Since Capgemini has a partnership with Mendix and not with Rollbase, the most straightforward

approach is to study IT management for applications that run on the Mendix platform. An additional advantage of studying Mendix management is that the link between management and software development can be made within Capgemini.

The global management framework is the set of processes that are relevant in a cloud computing environment. This set as well as the processes themselves should be based on experience and good practices to be effective. The input for developing a first concept of the SaaS framework will therefore come from the existing WoW framework for traditional applications, ITIL version 3, the ASL for SaaS proposition by van der Pols (2008) and the implications of the distinguishing characteristics of SaaS applications compared to traditional applications. Hevner, March, Park, & Ram (2004) mention five evaluation methods for the proposed design. The method used in this study will be *observational*. The framework theory will be incrementally built on what Hevner et al. (2004) refer to as 'the environment', justified by case/field studies. The case study approach itself will be based on the description by Yin (2003).The key subject of the case study interviews, observations and documents will be the practical usability of the conceptual framework that has been derived from theory. The interview reports and documentation will be analyzed using open and selective coding techniques that are common in quantitative research (Mohan, Xu, Cao, & Ramesh, 2008; Strauss & Corbin, 1998). The outcome will be integrated in the first concept to come to a second concept of the framework which will then be qualitatively evaluated by seeking feedback from a group of experienced practitioners. The review group will include application management specialists from PostNL and Capgemini. The review experts may have experience in traditional and/or SaaS application management. The comments of the reviewers will be processed and used for the final iteration in improving the concept to come to a final SaaS management framework. After both the case study and expert evaluation phases, the conclusions will describe the processes' validity and effectiveness. The same method as used for the framework will be used to incrementally develop the *configuration management* and *change management* processes, building from traditional process descriptions, scientific literature and the case studies. The process of developing, testing, improving and evaluating the framework and process descriptions is visualized in figure 5.

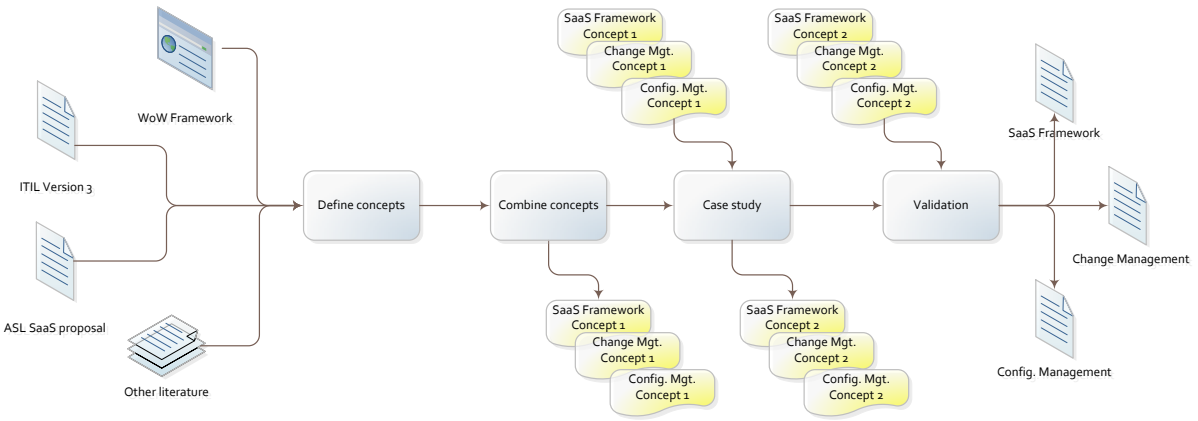


Figure 5: Development of the framework and two processes by a design science research approach

1.5. SCOPE AND LIMITATIONS

The time scope of the research is seven months. In this period, relevant literature will be studied, case studies will be conducted and experts in the management of SaaS solutions will be asked to review the artifacts. The results will be processed in a final SaaS Management Framework.

The initial case study has been limited to two selected case companies that deploy SaaS applications successfully developed with the model-driven development method on a supporting platform. The cases were selected by the following criteria: they need to be developed as SaaS on a platform that is offered by a partner of Capgemini and the development method of the service has to be model-driven. Cases that comply with the first criterion are SaaS services that have successfully been deployed on the Mendix, Be Informed, Salesforce or Microsoft Azure platforms. Of these, only Mendix currently provides a real model-driven development PaaS. The delimitation of this study is visually represented in figure 6. A case study may cover one or multiple applications within the two independent organizations. The goal of the case studies is twofold; on the one hand to test, improve and evaluate the global management framework and on the other hand to test, improve and

evaluate a process model for the processes *Configuration Management* and *Change Management* that are both part of the global SaaS management framework.

Cloud computing is a very broad concept and has therefore in this study been limited to the SaaS layer on top of the aforementioned platforms. The PaaS layer itself and layers of IaaS, BPaaS, HuaaS, etc. all fall outside the direct scope. Also preceding phases to management in both the software development and cloud lifecycle like 'requirements', 'definition', 'development' and 'testing' will not be considered within the context of this research.

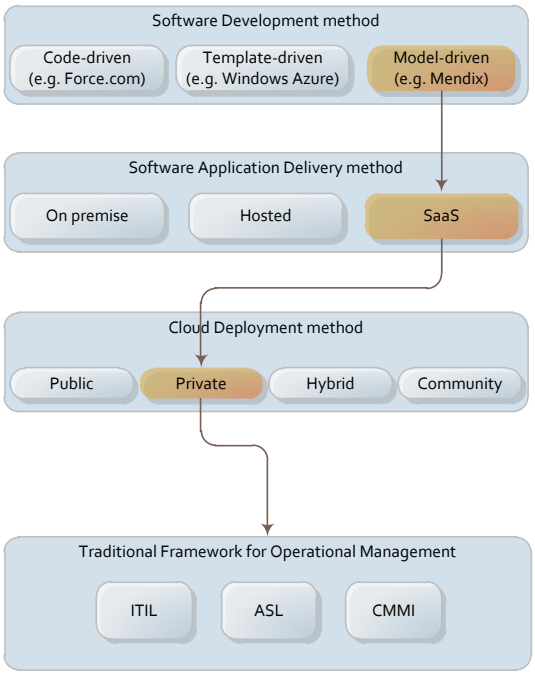


Figure 6: a visualization of the scope and goal of the research

With regard to IT management, a distinction can be made in strategic, tactical and operational management processes. Boynton, Zmud, & Jacobs (1994) use this distinction by referring to the IBM model for IT management when researching the influence of IT management practice. The current version of the IBM model, which is now called "Process Reference Model for IT Version 3" defines eight categories of processes that in total contain 309 activities (IBM, 2008). The IBM model is an extension of the ITIL framework, which is considered the de-facto standard. This research will focus on operational and tactical management processes and not on the strategic level, since the software deployment and delivery methods do not influence the process flow for strategic decision-making. In ITIL version 3, the tactical and operational processes are categorized in the "Service Design", "Service Transition" and "Service Operation" phases. In the IBM model this corresponds to the categories "Resilience", "Operations" and "Transition".

IT is used to working with quality and maturity defining models. In this research, the management processes will take the maturity levels and accompanying best practices and requirements of CMMI into account. Most cloud services will comply to CMMI level 4 (out of 5), meaning that detailed data on the software process and quality is being collected and there is agility and flexibility. Qualitative process management is in place and provides the necessary management steering information.

Management will be defined as the management, maintenance, and control of the services. Governance is considered to take place on a higher strategic level and does not play a role in this.

The management of SaaS solutions will be focused on the service functionalities. This means that management of data, hardware, infrastructure, people (skills), projects, processes, etc. will not be considered key in this research, but may be inherently connected.

1.6. THESIS STRUCTURE

This report starts chapter two with the relevant theoretical background on cloud computing, software development and IT management. The aim of this chapter is twofold; on the one hand to explore current literature on the subjects and on the other hand to define the meaning of the subjects in this study. The third chapter elaborates on the literature by presenting the first concept of a SaaS management framework and the configuration and change management processes.

The fourth and fifth chapters represent the practical part of the research, presenting the case study results and expert reviews. The framework incrementally becomes more mature in chapters three, four and five. The changes applied in each step of the research and their rationale is explained.

In chapter six, the conclusion elaborates on the final framework and preliminary iterations. The limitations of the framework are discussed and recommendations to Capgemini and customer organizations are being done. Also research validity, generalization and further research are discussed.

2. THEORETICAL FOUNDATION

A recent article on behalf of ABN AMRO (February 2012) shows that the percentage of companies that use cloud based services in the Netherlands has doubled from 20% in 2011 to 40% in 2012. This was concluded by a recent study performed by research company *Heliview Research*. The paper shows that cloud computing is quickly gaining ground due to its characteristics that fulfill the client's wish for flexibility, despite the doubts they may have about the safety of data (74% of surveyed), availability (61%) and continuity of the supplier (55%) of cloud computing (Heliview Research, 2012).

The business world is increasingly adapting cloud computing solutions. This 'new' technology is becoming significantly important to modern companies. The first paragraph of this chapter explores the different aspects and definition of cloud computing, how the technology emerged and in which forms it is being used nowadays. The second paragraph addresses different methods of software development and delivery and the third paragraph elaborates on the meaning of management, maintenance and governance of software applications. This structure addresses the three subjects that are important in this research: Cloud computing, Software development and Management.

2.1. PRESENTING THE CLOUD

The introduction above shows that modern companies have a need for flexible solutions and they find this more and more in services that are delivered by the cloud. However, what does this cloud computing really mean, how does it work and is it suitable for every type of software application?

2.1.1. DEFINITION

Merging the definitions of Armbrust et al., (2010) and Sultan (2011) gives that cloud computing can be defined as the on-demand provisioning of resources and services, including the hardware and software of distributed computers (in data centers and server farms) to store and provide these services over a networked medium. Rosenberg & Mateos (2011) add to this that in principle cloud computing makes pooled resources available to any subscribing user, hardware resource usage is maximized by virtualization, resources are dynamically scaled up and down when needed, virtual machines can be created, deployed and deleted in an automated manner and resource usage is billed on per-use basis. The final version of the U.S. National Institute of Standards and Technology (NIST) definition of cloud computing confirms the aforementioned characteristics and explains three different service models and four deployment models that are entailed by cloud computing (Mell & Grance, 2011).

The three service models or abstraction levels cloud computing can be divided into do somewhat overlap each other in the definitions used by different authors, but authors agree on dividing the services into three levels. The highest level is Software-as-a-Service (SaaS) and encompasses the end-user applications. The middle level is referred to as Platform-as-a-Service (PaaS) and can be compared to the abstraction level of the Operating System (OS) on a personal computer. The lowest level of service is called Infrastructure-as-a-Service (IaaS) and includes the physical infrastructure and the hardware in datacenters (N. A. Sultan, 2011).

Cloud services can also be distinguished in their deployment availability. When a cloud service is available to the general public on pay-as-you-go basis, it is called a *public cloud* service. Using such services is called *utility computing*. If the cloud service is not available to the general public but hosted in an internal datacenter environment that is only available within the organization, however being large enough to benefit from the advantages of cloud computing, it is called a *private cloud* service (Armbrust et al., 2010). Combinations of public and private clouds are also possible. This form is then called a *hybrid cloud*. Typically companies outsource non-critical information to public cloud services and keep the critical information in which the company distinguishes itself from other organizations controlled within the own company (Marston, Li, Bandyopadhyay, Zhang, & Ghalsasi, 2011). However, all kinds of variations can be realized. The fourth and last deployment method is denoted *community cloud*. The community is created by using the resources that are free on user computers. The nodes fulfill all roles (consumer, producer and coordinator) themselves. The advantage of a community cloud is mainly in the structure which does not lay all control at a single vendor. Furthermore it is also more environmentally sustainable since it makes use of existing unused computing

resources (Marinos & Briscoe, 2009). Sultan (2011) states that the name “cloud” was probably inspired by IT models and text books which commonly picture a remote network or the internet as a cloud.

Cloud applications have some distinguishing characteristics with respect to traditional software. Gong, Liu, Zhang, Chen, & Gong (2010) describe four characteristics. The first characteristic is conceptual: the cloud is service oriented, which means that software is abstract and accessible. The second and third characteristics are technical: loose coupling and strong fault tolerance. Loose coupling encompasses virtualization and other technologies to separate the infrastructure in logical and physical. Strong fault tolerance means that computation states are periodically saved on stable storage to provide a rollback possibility when an error occurs. The fourth characteristic is economic and relates to the business model, where pay-per-use is the most favorite, against sole upfront capital expenditure. The fifth characteristic entails user experience: cloud services are easy to use. The cloud model reduces information technology overhead for the end-user. Other characteristics are: TCP/IP based, virtualization and high security (Gong et al., 2010). Surgient (2009) gives another classification of characteristics, defining the following five characteristics of cloud computing: Dynamic computing infrastructure, IT service-centric approach, Self-service based usage model, Minimally or self-managed platform and Consumption-based billing.

2.1.2. KEY ADVANTAGES

In comparison to traditional software that is delivered on premise and installed locally, cloud computing has two core advantages. First, there is the advantage of scalability. Resources become immediately available when requested and second, the customer only has to pay for the resources he actually needs. This pay-per-use model makes a huge initial investment (capital expenditure) in resources obsolete and creates significant cost advantages (Lin, Fu, Zhu, & Dasmalchi, 2009). Since the cost of resources has become an operational expenditure, it can be managed. This creates a situation in which customers can easily try out software services and just stop using them when they no longer need the application or when the solution does not seem to work as good as thought. There will have been negligible start-up costs and when finished, all costs will cease to exist. This is the flexibility that e.g. project teams and companies need when trying out new ideas or to easily scale existing applications. Next to these two main advantages, Marston et al. (2011) mention three more advantages: almost immediate access to hardware resources, lower IT barriers to innovation, possibilities to create new classes of applications that were not possible before.

Cloud solutions are hosted on-line and available wherever someone has access to the internet or intranet. This results in the great advantage that the work can actually be done wherever the work happens, meaning that e.g. a policeman can do the ticket administration at the streets and a doctor or nurse can immediately update the patient’s status administration at the bed. For the efficiency and continuity of the business, it is of great importance that administration is not separated from the process activities by doing it in a later stadium, but that it is integrated in the physical activity.

2.1.3. ENTRANCE BARRIERS AND CHALLENGES

The main barriers for companies to start using cloud computing are concerns about their privacy and about privacy legislation that differs between countries where the data may actually be stored. The cloud computing provider itself may be trusted, but the stored data may under certain laws be claimed by the government of the country where the data are physically stored. Also, the EU prohibits the processing and storage of personal data. If this is an issue, the customer should explore upfront in which locations the data of a cloud provider can possibly be physically stored (Leavitt, 2009). The second main-issue is vendor lock-in, since no system really supports easy transfer and changeover to another system (N. Sultan, 2010).

Armbrust et al. (2010) present a top-10 of obstacles and matching opportunities in the use of cloud computing. Their summary gives an overview of the most frequent concerns companies have against the use of cloud computing and shows realistic solutions. It is adopted for its informational value in table 1.

Obstacle	Opportunity
1 Availability/Business Continuity	Use Multiple Cloud Providers
2 Data Lock-In	Standardize APIs; Compatible SW to enable Surge or Hybrid Cloud Computing
3 Data Confidentiality and Auditability	Deploy Encryption, VLANs, Firewalls
4 Data Transfer Bottlenecks	FedExing Disks; Higher BW Switches
5 Performance Unpredictability	Improved VM Support; Flash Memory; Gang Schedule VMs
6 Scalable Storage	Invent Scalable Store
7 Bugs in Large Distributed Systems	Invent Debugger that relies on Distributed VMs
8 Scaling Quickly	Invent Auto-scaler that relies on ML; Snapshots for Conservation
9 Reputation Fate Sharing	Offer reputation-guarding services like those for email
10 Software Licensing	Pay-for-use licenses

Table 1: Top-10 cloud computing obstacles (Armbrust et al., 2010)

2.1.4. THE CLOUD COMPUTING STACK

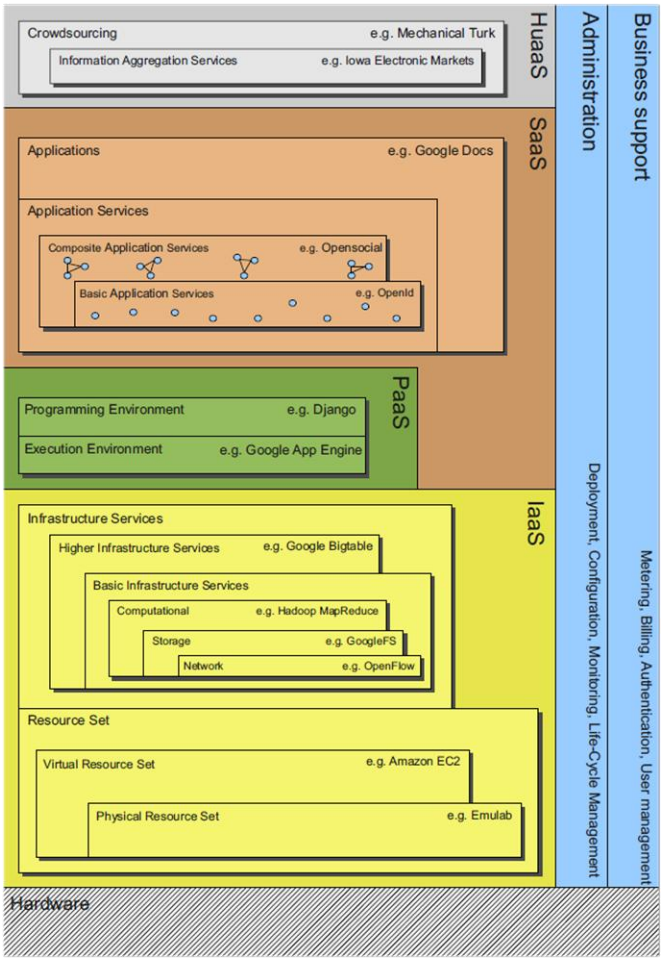


Figure 7: Cloud Stack (Lenk et al., 2009)

Authors generally agree that cloud computing distinguishes different levels of services (e.g. Marinos & Briscoe, 2009; Miller & Veiga, 2009; Sultan, 2011). However, some would rather not use the popular terms for it, since vendor's definitions still vary (Armbrust et al., 2010). All the levels of service together form a stack. Cloud service providers provide services in one or more levels of this stack. A high-level service will inevitably need the lower-level services to build upon.

Lenk, Klems, Nimis, Tai, & Sandholm (2009) present a very clear overview of the cloud stack (figure 7). The service stack's foundation is at the hardware and infrastructure. Examples of IaaS services are Amazon EC2, GoGrid, Rackspace and Nirvanix. If a customer does not only need hardware, but also a common platform to build and run applications on, the IaaS can be extended with a PaaS. Examples of platforms are Google App Engine, Microsoft Azure, VMWare Cloud Foundry, Salesforce Heroku, Red Hat OpenShift and Cloudbees.

The aforementioned two levels are typically interesting to SaaS service providers. They develop their services on a platform and offer them as a SaaS application. The end-users will just use the SaaS layer providing them with software on-demand, but actually the PaaS and IaaS layers are working in the background of this abstraction layer. Examples of SaaS applications are salesforce.com, Yahoo! Mail, Hotmail, Gmail and Google Search.

To be complete, Lenk et al., (2009) also show Human-as-a-Service (HuaaS) in the stack representation model. The human can be seen as an extra layer of service provisioning, e.g. in the case of crowdsourcing. Despite the fact that this appliance can definitely be seen as an extra layer and is therefore worth mentioning, it is not really being adopted by other authors. A more accepted layer in the literature is Business Process-as-a-Service (BPaaS), which provides the coupling and integration of multiple applications, resulting in a well-integrated ecosystem that provides users with an end-to-end business process (Papazoglou & van den Heuvel, 2011). Well-integrated business processes will reduce cost and decrease throughput time. A company in this market is Cordys, a Dutch organization that offers functionalities and applications that are based on their PaaS (Cordys B.V., 2012). The principle of the cloud service-stack can be extended with many other services that satisfy the basic characteristics of cloud computing. Chang, Walters, & Wills (2012) e.g. also mention BPaaS (although denoted BIaS, Business Integration as a Service) and argue the use of RAaaS (Risk Analysis as a Service) and Return on Investment (ROI) Measurement as a Service (RMaaS).

This research, however, will consider SaaS as the ultimate top layer in the cloud stack, since extended layers can actually perform management tasks, but are irrelevant in the context of development and defining the operational management of cloud services. The SaaS services are typically developed and used in a generally applicable standard lifecycle. This will be clarified in the next section.

2.1.5. CLOUD SERVICE ENVIRONMENT AND LIFECYCLE

The cloud environment has different actors, all with a different view on cloud computing. First, the owner of the computing environment, the Application Service Provider (ASP), sees it as an environment to manage computing resources of SaaS applications. Second, the developer sees it as a set of development tools to develop SaaS applications in an integrated environment. Third, to service providers it represents a hosted environment to present their application services, and fourth, to customers/consumers it represents a virtual market to find, purchase and use online application services (Tang, Zhang, & Jiang, 2010). A cloud administrator monitors and manages the services and resources. These roles are in line with the roles and responsibilities defined in the classical Service Oriented Architecture (SOA) (Breiter & Behrendt, 2009). Customers are both individual users and organizations with multiple end users that subscribe to a web application or service. Tang et al. (2010) propose two different business models based on this ecosystem: *platform centric* and *service provider centric*. In the first model the platform has a brokering function between customers and service providers which provide multiple services. The service provider centric model lets customers interact with providers directly. The provider will then connect a service to a customer. The advantage of the second model is a direct cash flow from customer to service provider. A disadvantage is that a single service provider is not able to offer a catalogue of services as large as an operation platform can, since the platform typically offers services from multiple providers.

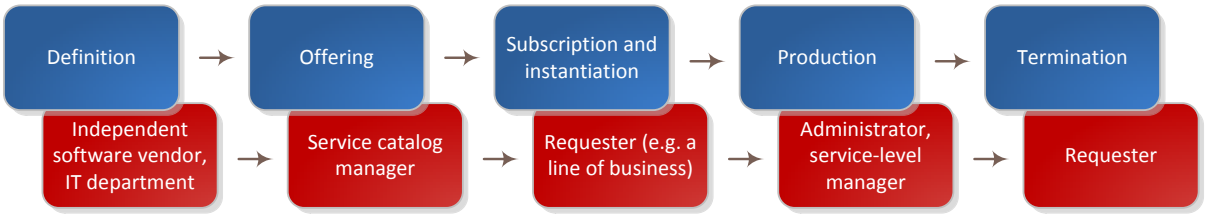


Figure 8: Lifecycle of a cloud service with the responsible actors associated (Breiter & Behrendt, 2009)

Customers, both organizations and end users, will only perceive the software solution they are using a SaaS, the underlying functional PaaS and IaaS services are concealed. However, since they actually use multiple underlying services, the customer could have contractual arrangements with multiple supplying parties. The PaaS or SaaS provider, whichever is centric in the business model, should ideally reduce this complexity and provide the customer with a single support channel and a single transparent invoice (Beimborn, Miletzki, & Wenzel, 2011). Developers of the consumer services work at the lower PaaS level of abstraction and the ASP works at the lowest IaaS level of abstraction. They all have their own viewing point and responsibilities in the cloud computing lifecycle.

The general lifecycle of cloud services in figure 8 describes five global steps that represent a formal way of working. Already in the first step all management knowledge is captured in a service template. This contains all knowledge that is needed to create the cloud service instance and to manage it. The service template contains abstract build and management plans which define the structure and operations of the service without bothering the administrator with information about lower-level resources. The service template is a universal definition that does not contain detailed information like pricing and technical resources. It will be adjusted to the specific needs of the customer in the offering phase. The subscription and instantiation phase is about publishing the service in the service catalog. When a requester selects an offering and sets the parameters like capacity, availability, performance and duration, the application is actually being built on a location in the datacenter. In the production phase, subscribers can start using the service. Breiter & Behrendt (2009) state that the management of instances involves two aspects: autonomic management that ensures SLA's (Software License Agreements) are being met and management operations that are manually executed by the administrator. All possible management actions are defined in the cloud service template and can be reduced to the specific needs of the service requester in the offering phase. The cloud service instance is being terminated when it is no longer needed or when the resource reservation expires. All allocated resources then go back to their pools and become available to other services (Breiter & Behrendt, 2009).

2.1.6. A PART OF EVOLUTION HISTORY

Larman & Basili (2003) argue that iterative and incremental development started with Walter Shewhart's work in the 1930s. Shewhart was a quality expert at Bell Labs who proposed a series of plan-do-study-act (PDSA) cycles for quality improvement. W. Edwards Deming began promoting the PDSA cycle from the 1940s and from the early 1960s this principle was adopted in software development, starting with the NASA's Project Mercury. In this first project, time-boxed iterations of half a day were used, which means that two improved versions would be developed and tested/deployed every day. Royce (1970) was the first to propose a development method that would later be known as the waterfall method. His publication prescribed a strict activity sequence of requirements, analysis, design, coding, testing and operations. The author describes that the focus of this method was on the development of large computer programs that are to be delivered to a customer. Larman & Basili (2003) state that the waterfall method has long been considered the standard method for software development, however in the 1990s, iterative and incremental development became more accepted and popular and a lot of different software development methods as we know them today are based on the incremental principle. Examples of these software development processes are Rapid Application Development (RAD), Dynamic Systems Development Method (DSDM), Extreme Programming (XP), Feature-Driven Development (FDD), Agile and Scrum (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). The main advantage of these iterative methods is that they are more flexible than the 'standard' waterfall method. Flexibility is especially convenient when software requirements are subject to change or are likely to be refined during the development process.

Besides software development, also software delivery has had several different waves in which a specific method was leading. This started with the mainframe era, followed by the personal computer (PC) era that went through a transition with distributed computing (DC) to the ubiquitous computing (UC) era. The *mainframe era* began in the late 1950s. Large computer systems were established and kept evolving (Ebbers, O'Brien, & Ogden, 2006). Mainframes were mostly operated by experts behind closed doors and the computer would therefore be shared with many people. The applications and operating system were very specific and many companies who used a mainframe built their own OS. In 1984 the number of people using a personal computer first outgrew the number of people using shared computers; the *personal computer era* had emerged. Characteristic to a PC is that it is operated by a single person who stores personal files and applications on it. The internet started to grow and certain manufacturers started building large computer systems for connecting multiple processes within the company. This meant that people needed to start working together on a single system, which was the start of a transition called *distributed computing*. Client-server applications ran on multiple systems (a server and multiple clients) and the internet contained data that are being served to a client computer by other computers. This epoch of DC may also be called 'internet' or 'web' era and combines the former two era's. The applications that support company-wide business functions and integrate internal processes are nowadays called Enterprise Resource Planning (ERP) Systems. The cross-over to the *ubiquitous computing era* was not as clear as former transitions. Weiser & Brown (1996) claimed that this era would gradually emerge between the years of 2005 and 2020. The authors were right; the evolution in computing is currently all about ubiquity, including mobile and smart devices that are connected to

the internet. Extended ERP applications provide a firm with inter-organizational collaboration with suppliers and customers, which is characteristic for the UC wave. Also, more applications are being built in a modular fashion for better integration with software from different manufacturers. This gives users the option to select the best module for their purpose instead of being bound to a system from a single vendor. The UC era includes terms like 'web 2.0', 'web services' and 'cloud computing'. Some cloud services provide modular extension to existing data-structures and some cloud platforms support transformation of existing applications (Weiser & Brown, 1996). The mainframe, however, lives on, still resonating in the succeeding structures of computing in forms of client/server and client/cloud server (Hemsoth, 2010; Waters, 2005).

The different eras had significant consequences for the way companies work and organize their IT expenses. Traditionally an organization would make large capital investments upfront to purchase resources like hardware, software, data storage and IT skills. Later, companies started to offer outsourcing of IT resources, providing organizations with the option to rent resources under contracts of several years. These contracts did not provide customers with much flexibility and were therefore not convenient in many situations. Cloud computing solutions emerged and made resources really flexible. A customer would only 'rent' and be charged for the resources that are actually considered necessary, which may fluctuate over time. When an organization would choose this approach, there is no need for capital expenditures, the expenditures have been operationalized (Loh & Venkatraman, 1992; Marston et al., 2011).

2.2. SOFTWARE DEVELOPMENT AND DELIVERY

Native or traditional software are the terms that will be used in this paper to indicate the conventional type of software which needs to be installed on a personal computer or server as opposed to web applications that can mostly be accessed from within any web browser. SaaS software can generally be run inside the web browser on an external server, and therefore belongs to the category of web applications or web services. This chapter will review the traditional development methods to produce native software applications, but it will particularly elaborate on modern cloud related development and delivery models; e.g. the agile and model-driven development methods that are being used in the context of this research.

Software development initially started using the waterfall method. This method for software development was first composed by Royce (1970) and has not changed much ever since. For a long time it was seen as the best method for structuring the overall software development process and it is still being used and very efficient in development projects where the final requirements can be defined upfront and are not likely to change during the development process. However, IT requirements as well as IT capabilities develop quickly and in most cases it will be hard to deliver a product at the end of a lengthy project that still satisfies the exact needs. In this case it will be convenient to use an agile iterative and incremental building process like Scrum. The agile methods do not focus on satisfying the customer at the time of forming the requirements, but at the time of delivering the project's end product (Highsmith & Cockburn, 2001). The third method follows up on Agile by providing a hosted platform in the cloud that can be used to develop and deploy applications and making it immediately available to testers and end-users. Since it is cloud technology, this will be provided to users via the internet, likely in the browser. However, in a sense, PaaS actually is a new development method, since developers do not need to configure development and test servers and they do not need to care about storage, security, OS and server patches. This makes the development method more productive and reduces the development cost (Lawton, 2008).

Software development methods are closely connected to the methods for software delivery. As mentioned above, PaaS development is a real convenient enabler of Agile and makes the functionality in the same way directly available to end-users, as opposed to the waterfall method and local software installation.

2.2.1. SOFTWARE DELIVERY MODELS

There are three major methods for software delivery. These methods have all experienced their own times of popularity, as can be read in paragraph 2.1.6 (a part of evolution history). Some authors have split the delivery models into more than the three models that will be briefly discussed below, but in general delivery is always reducible to these three methods (Diversity Analysis, 2010).

On-premise
The on-premise delivery is the most traditional way of software deployment. In a time where there was no internet, all systems had to be run locally at the customer's site. The manufacturer develops a software application and the software licensee deploys it on his own hardware at the desktop computers or datacenter on his premise (Diversity Analysis, 2010; Sun, Zhang, Chen, Zhang, & Liang, 2007).

Hosted
In the off-site hosting delivery method, the provider owns the hardware and provides the same software as in the on-premise method, but via the internet. The software is not physically present at the customer's premise. A hosting provider can offer the same software to multiple companies, using a single set of hardware to exploit more of the hardware capacity. An offshoot of hosted delivery is the Application Service Provider (ASP). The advantage for the customer is that the overhead costs are transferred to the provider (Diversity Analysis, 2010; Waters, 2005).

SaaS
SaaS has comparisons with hosted software, but differs in the aspect that it is designed to be fully on-demand, not based on contracts of several years. This on-demand delivery can be provided from different locations, all deploying the same kind of software but in a different environment. The environments are called public cloud, private cloud, hybrid cloud and community cloud, like explained in the cloud definition paragraph (Diversity Analysis, 2010; Sun et al., 2007).

In theory any development method can be combined with any delivery method, but some combinations support each other better than others. SaaS solutions for example can easily be developed using an agile development method, since users can connect to the application immediately without first having to compile and install the software with all its additional drawbacks. Agile plays a big role in the SaaS services that are subject to this research and will therefore be explained shortly in the following section.

2.2.2. AGILE DEVELOPMENT

Agile has become a major concept in software development. It implies effectiveness and maneuverability. Expectations of the business and market demands have grown over the years and Agile is a method to meet these demands and expectations by collaborating with the customer in the development process (Cockburn, 2001; Highsmith & Cockburn, 2001). When using the waterfall method, the company of the end-users would represent the customers and function as a 'filter' between the supplier and the users. With agile methods, the end-users themselves get directly in contact with the product that is being developed and they can participate in it. SaaS deployment models provide direct availability to the end-user, who can test it and comment on it, thus shifting the power from the intermediate organizations to the end-users. The advantage of agile systems development is in moving away from 'introverted' development by involving the end-users, resulting in a more innovative and valuable system (Conboy & Morgan, 2011).

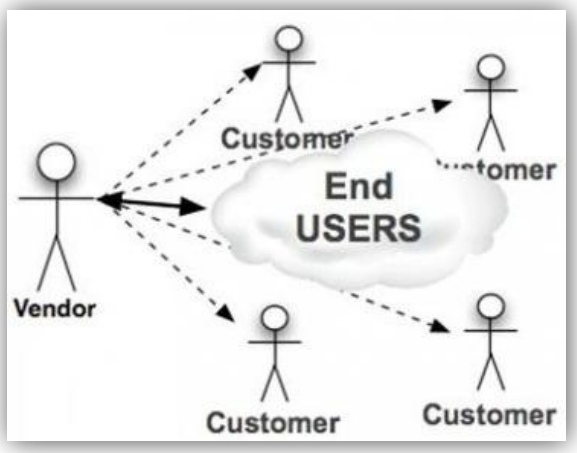


Figure 9: SaaS Business Model (source: www.squidoo.com/SaaSDevelopment)

Pressman (2009) determines four underlying values that prelude the establishment of better ways to develop software: individuals and interactions are more important than processes and tools, working software is more important than comprehensive documentation, user collaboration is more important than contract negotiation and preferring responding to change rather than following a plan. These values make it possible to respond effectively to changes, it creates usable communication between different stakeholders and it makes users part of a team that is organized and in control of the work. This all means that software can be developed and delivered in a rapid and incremental fashion. An agile way of working saves costs since the end product fits the business better and changes are even possible in a relatively late development stage against lower cost than when using the waterfall method (Pressman, 2009). This, however, does not mean total anarchy, since the

participants are in control of the project as a team. An architect can be part of this team or working closely together with the team to drive them in the direction of good architecture, focusing on aspects like the technology stack, design patterns and quality attributes (Madison, 2010).

2.2.3. MODEL-DRIVEN APPLICATION DEVELOPMENT

The companies Mendix and Be Informed, mentioned in the introduction chapter, as well as the by Lawton (2008) researched software from Rollbase (Rollbase Inc., 2012) offer platforms that specifically provide model-driven development tools which contain prebuilt software components. Model-Driven Engineering (MDE) is an old subject in IT literature, but many difficulties made that it never really came off the ground in practice. Its predecessor Computer-Aided Software Engineering (CASE) emerged in the early 1980s (McKinney, 1991; Schmidt, 2006). Model-Driven Architecture (MDA) was first formalized by the Object Management Group (OMG) in a 2001 document (Miller & Mukerji, 2001), but was still in development at that time. The OMG is the developer and maintainer of the widely used Business Process Model and Notation (BPMN), a standard for business process modeling. This group currently defines MDA as:

"*An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform*" (Object Management Group, 2007).

Kent (2006) states that MDA is the concept of models playing a direct role in software production. MDE complements this with a combination of process and analysis. A definition of this concept is as follows:

"*MDE is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed to concrete implementations*" (France & Rumpe, 2007).

MDE can be seen as a fifth-generation programming language since it is based on solving problems by solely defining constraints to the program, rather than a programmer writing an algorithm. When no classic coding is necessary, the only necessity for building and extending programs is knowledge of the tool. It removes the complexity from every phase in the development process by taking care of the algorithms in the background and therefore creating a higher level of abstraction for the application developer. A higher level of abstraction implies that there is no need to refer to underlying structures and technologies, developers get to focus on system-wide correctness and performance instead of programmatical details (Schmidt, 2006). In this next-generation or higher abstraction level programming languages, preceding generations still reside and function in the background, like a stack. Every new language generation made application development a bit more abstract and better understandable for humans, but in the end every computer program must be converted to pure binary machine code for the computer to be understood. Machine code is seen as the first-generation programming language or the bottom of the stack. The second-generation language improved human understanding by providing the programmer with an assembler for converting symbolic statements to binary machine code. This is called assembly language. The third-generation is called a "high-level" programming language since it makes expressions more like the normal mode of expression for humans, e.g. algebraic expressions. High-level programming languages are hardware-independent. The computer characteristics no longer influence the way an algorithm should be expressed and a compiler is needed to convert the statements to machine code. Fourth-generation languages are specifically designed to be closer to natural human languages. For example database languages belong to this category. They are designed to enhance usability and reduce programming effort. Fifth-generation languages can be defined by development with a visual or graphical interface to create source language as opposed to a text editor (Balasubramanian, Gokhale, Karsai, Sztipanovits, & Neema, 2006; Henry, 2007; Louden & Lambert, 2012).

With this information on visual programming; the fifth generation programming language, it does make sense that different authors define different distinctions between SaaS and PaaS. Model-driven development in this case provides a visual development environment as part of the PaaS to create SaaS applications by a visual model. On the platform, the development interface operates next to the usage interface. The development tool is part of the PaaS, but can also be seen as a service in itself. To the actual SaaS users, only the end-service is visible and the underlying technologies remain concealed. However, a SaaS user could in theory also be a developer and have access to the modeling tool (service).

In this research, the software from Mendix will be explored when considering MDE, since Capgemini uses this software to develop in a model-driven way. The Mendix tool exists of four parts: the business modeler, the model repository, the business server and the rich client. The business modeler is used to create the system, supporting three main model types to do so (Henkel & Stirna, 2010):

- Meta-model: model of the information structure of the system
- Form-model: the user interface which is built from menus and forms
- Microflow: special procedural logic. Similar to UML Activity diagram, but with some extensions

The advantage of modeler software is that it conceals the underlying technical difficulties. The human modeler does not have to think about the technical design of the system, only about the functional design. The software will automatically create and at the same time check the technical integrity of the model and raise a warning when something is incorrect. Other advantages are that business analysts can take actively part in the development cycle, and the flexibility to change the application in less time than needed for traditional development. On top of that, Mendix maintains a library of process templates and applications, which may make parts of business analyst and developer efforts no longer necessary (Mendix Technology B.V., 2011).

2.2.4. TEMPLATE-DRIVEN APPLICATION DEVELOPMENT

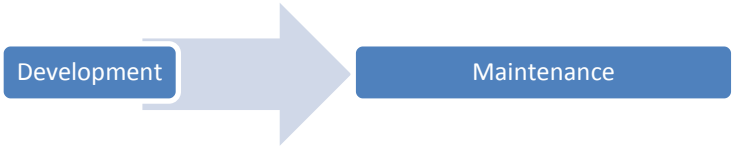
The use of templates is another modern approach of software development. Papazoglou & van den Heuvel (2011) state that next to a model-driven approach, metadata constructs (called templates) are used to provide an understanding of the features of a cloud service for the purpose of management of the service. Stahl, Völter, & Czarnecki (2006) describe that MDE is actually a higher abstraction of the template-based metamodel. The graphically designed model is transformed to programming code by use of code generation templates, directed by a Domain-Specific Language (DSL) metamodel.

Since all lower-level transformation steps of the visual model are automated in the case of MDE, a developer or development tool can also be tuned to one of these lower abstraction levels. This is for example the case when a developer manually adds code to an application that has been developed using MDE. Another appliance of development on a lower-level abstraction is when the development is tuned to an abstraction layer between the visual modeling and code writing. E.g. when developing using the metamodel and code templates, without developing in a 'drag-and-drop' graphical interface (Stahl et al., 2006).

Louridas (2010) states that the Force.com platform supports a programming framework for the metadata-driven development model. Windows Azure provides the .NET framework which includes a library. A library can in a way be seen as templates, since it means a collection of resources that can be used to develop software. This can include pre-written code, subroutines, classes, values and type specifications.

2.3. TRADITIONAL IT MANAGEMENT

The emphasis in the software lifecycle used to be more on development than on the phase where the system is live and maintenance, management and systems enhancement takes place. This disproportion is potentially dangerous, since most costs occur in the maintenance stage (table 2).



Attention	Much	Little
Costs	20%	80%
Duration	Up to 3 years	8 to 30 years
Users	Not yet	Yes
Management	Project	Department

Table 2: Comparison between development and maintenance (van der Pols & Backer, 2006)

Traditional IT management processes are defined by standardized methods such as ITIL, ASL and BiSL, which are based on best-practices to provide a framework for organizing the management processes. These traditional methods capture the management functions in processes and the complete set of processes is called the management framework. Other methods have been developed to automate and simplify these service processes, like the IBM Service Management Architecture (Lindquist, Madduri, Paul, & Rajaraman, 2007). Papazoglou & van den Heuvel (2011) state that developers usually approach representation and management of cloud resources model-driven or with metadata constructs, called templates. The authors propose a new approach called cloud blueprinting. The blueprinting approach sees all cloud-stack layers as general purpose commodities that should all be able to be chosen individually, instead of not knowing anything about the background PaaS and IaaS layers when using a SaaS. This should minimize dependencies and vendor lock-in (4CaaSt, 2011). However, these three methods emphasize a developer point of view, in which context management is particularly about connectability and understanding of the service's features. Management is in this thesis about operational service delivery.

Nowadays, management and maintenance of information systems has gained more emphasis from organizations instead of focusing mainly on the development of systems. There is awareness that the management of a system is very important to guarantee the continuity of the hard- and software and moreover for continuous alignment with the business environment after the development phase. This necessity exists because business requirements change over time and the system needs to support the business operations. That organizations should find this important appears from Gartner research that shows that 70-80% of the overall time and cost goes to the operational phase in the lifecycle of IT products (van Bon et al., 2007). IT Service Management (ITSM) is defined by van Bon et al. (2007) as: "*the management of all processes that co-operate to ensure the quality of live IT services, according to the levels of service agreed with the customer*". The scope of ITSM is the facilitating technology, not the business use or information to design and control. The activities and decisions take place on strategic, tactical and operational business levels.

2.3.1. DEFINING GOVERNANCE, MANAGEMENT, MAINTENANCE & CONTROL

Literature distinguishes between IT Governance and IT Management. Shaw, Chung, Cheng, & Fu (2012) state that corporate governance is about the supervision of management-level processes, structures and relationships and that corporate management includes the operations necessary to realize manager goals. In IT, these terms parallel the business. IT management is therefore the operationalization that is shaped by the principles and goals of IT governance. The strategic alignment model as proposed by Henderson & Venkatraman (1993) confirms this strategic and operational interconnection between the business and IT (figure 10).

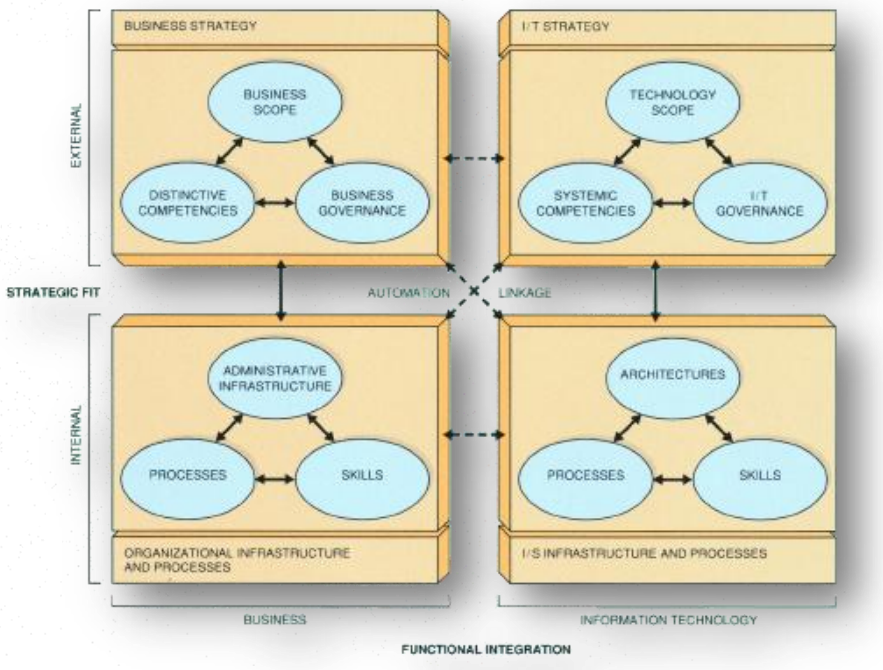


Figure 10: Strategic alignment model (Henderson & Venkatraman, 1993)

Not only IT Governance and IT Management will be defined here, also IT Control and IT Maintenance are subjects that are associated with IT management and need to be introduced since they are important elements of both governance and management.

IT Governance is a broad concept that includes different

aspects. Sallé (2004) and de Haes & van Grembergen (2004) mention in their definition that IT governance is the responsibility of board, executive management and IT management and that it is about the formulation and implementation of IT strategy. Important is that, since it is the responsibility of business management, a good integration of business and IT should be achieved. The authors mention the strategic alignment model and the balanced scorecard as tools for strategic alignment. Service Level Agreements and the maturity models for all COBIT's 34 processes play a role in defining the business' expectations of IT. De Haes & van Grembergen (2004) emphasize that IT governance is about a better fit between business and IT and improving the quality of information services. Simonsson & Johnson (2006) complement this with their findings that IT governance is not only about strategic IT goals, but also about processes, technology and personnel. The decision-making process involves understanding, deciding and monitoring. Even the scope is broader; not only strategic long-term, but also tactical level short-term decision-making is part of IT governance. Webb, Pollard, & Ridley (2006) describe that IT governance should include five elements: strategic alignment, delivery of business value through IT, performance management, risk management and control and accountability. In conclusion of all former authors, this research will define IT governance as the processes in the strategic and tactical levels of management to gain alignment between business and IT and improve the maturity of IT services.

IT Management is another broad term of disciplines and is in many aspects entangled with IT Governance. McNurlin, Sprague, & Bui (2009) state that IT management is about managing the technological resources in accordance with its needs and priorities, including functions like budgeting, staffing, organizing and controlling. Since no generally accepted standard definition of IT management can be found, the distinction between governance and management is sometimes blurred or the terms are interchanged. Management has multiple aspects; the first is that IT management mainly takes place at the tactical and operational level business layers. The second is that it is the next phase after software development, meaning that the first phase (service development) is about project delivery and IT management follows up with continuity services. Schwalbe (2011) states that the generic term "systems management" addresses business, technological and organizational aspects that are associated with the creation, maintaining and making changes to a system. Breiter & Behrendt (2009) and Iwasa (2011) use a model with three different categories of IT management. The first is denoted *service management* and includes the operational IT management processes as described by the ITIL framework. The second is denoted *system management* and is about monitoring events, end-use performance and automated operations of systems. The third is *resource management* and is about the management of physical resources and operating systems/middleware. Peterson (2003) describes the differences between IT Service Management and IT Governance as follows: "*Whereas the domain of IT management focuses on the efficient and effective supply of IT services and products, and the management of IT operations, IT Governance faces the dual demand of (1) contributing to present business operations and performance, and (2) transforming and positioning IT for meeting future business challenges*". This means that IT Governance has a more external, future-oriented focus and IT management is more internal-focused and oriented to the present. This thesis will define IT management as the tactical and operational business, technological and organizational issues concerned with creating, maintaining and change-making to IT services.

IT Control is about the steering of processes in the organization by means of performance indicators and standards (Boland, 1979). Internal control is an important part of this, performed not only by the board or management, but also by other personnel. It is about ensuring a degree of certainty in reaching the organization's objectives. The COSO (1992) internal control integrated framework defines four categories: effectiveness and efficiency of operations, reliability of financial reporting, compliance with applicable laws and regulations and safeguarding the organization's assets. IT control will be defined as measuring the level of process performance against a given standard and the ability to adjust and improve the processes to comply with the standard. IT control is not a direct subject to this thesis, but may appear as part of IT governance and IT management.

IT Maintenance can be divided into multiple categories, e.g. software maintenance, hardware and infrastructure maintenance and skills maintenance (training people). Since this research is about the software services layer of cloud computing, only software maintenance will be considered relevant and therefore be defined. Midha & Bhattacherjee (2012) state that software maintenance is about the way of managing developers, since they are the one critical resource in software maintenance projects. Specifically it is about managing their participation and the allocation of responsibilities. The authors adopt three types of software maintenance: corrective, adaptive and perfective. Defining corrective maintenance as bug-fixing, adaptive as adding new functionalities and capabilities and perfective as being solely about enhancing the performance.

These three types were extended by a fourth type "preventive maintenance" by the original defining author (Lientz & Swanson, 1980), but was not adopted by Midha & Bhattacherjee because this type accounts for less than 4% of total maintenance tasks, and it can be considered mutually exclusive and exhaustive with the first three types. April, Huffman Hayes, Abran, & Dumke (2005) explain that the context of software maintenance involves customers and users, the infrastructure and operations department, developers, suppliers and upfront maintenance and the helpdesk. If an agile development method is used, software maintenance is included in the continuous lifecycle of development and usage.

When operationalizing the concept of IT management as described formerly, the ITSM method provides directives. Both the ITIL and ASL frameworks are generally mentioned when discussing IT governance models (Larsen, Pedersen, & Viborg Andersen, 2006), since they do include processes to cover all three business levels of strategic, tactical and operational (Souer, Honders, Versendaal, & Brinkkemper, 2008). However, interest in this thesis is in the operational implementation of IT management and therefore only the operational processes will be considered.

2.3.2. IT SERVICE MANAGEMENT, ITIL & ASL

Sallé (2004) describes three levels for the maturity of the IT organization's functioning. The levels can be seen as a stack in which the lower level(s) are necessary to support the upper level(s). The lowest level is IT Infrastructure Management (ITIM), which focuses on improving the management of the enterprise infrastructure in order to maximize the return on computing assets and on taking control of the infrastructure, devices and data. The middle level is IT Service Management (ITSM) and is about identifying the services the IT organization's customers need and focuses on planning and delivering those. Important management and planning aspects to the services are: availability, performance and security requirements. Furthermore, this level is also about managing SLA's to meet the agreed quality and cost targets. The IT organization reaches the top level and evolves to IT business value management (which is IT Governance) when the IT processes are completely integrated with the business lifecycle of service quality improvement and business agility. A simple schematic representation of the relationship between IT governance and IT management is shown in figure 11.

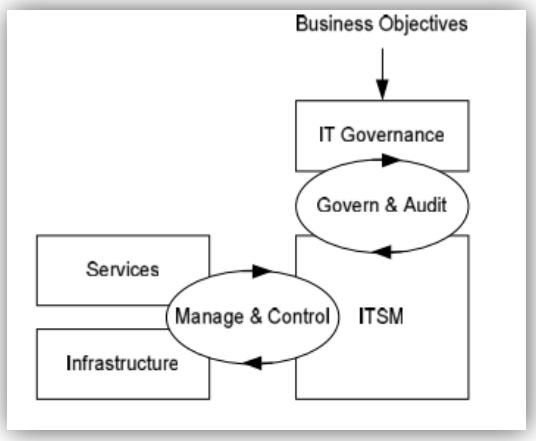


Figure 11: model between IT governance, IT service management and IT operations and services (Sallé, 2004)

ITSM sets the central focus to the IT services and is commonly defined as:

"*a set of processes that cooperate to ensure the quality of live IT services, according to the levels of service agreed to by the customer. It is superimposed on management domains such as systems management, network management, systems development, and on many process domains like change management, asset management and problem management*" (Young, 2004).

Another definition describes ITSM in a higher level of abstraction:

"*Service Management is a set of specialized organizational capabilities for providing value to customers in the form of services*" (Cartlidge et al., 2007).

Gartner analysis shows that ITSM is about optimizing the outcome of a service, not to actually process the outcome. In other terms, it must be seen as a guideline for improvement, not as a specific fixed set of rules or a goal in itself. Within ITSM, there are four key frameworks that define the anatomy of the service value chain. The goal of this chain is to regulate the process of developing, offering and maintaining services by first defining them in a service portfolio that is in agreement with

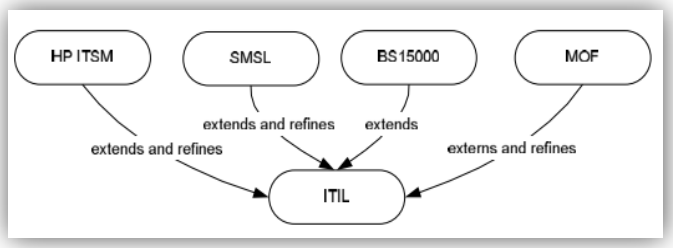


Figure 12: Other ITSM frameworks related to ITIL (Sallé, 2004)

the business or client organization, only after that they can be catalogued, executed and administered (Young, 2011).

Supporting ITSM, ITIL is the worldwide de-facto standard in Service Management (Behr, Kim, & Spafford, 2004; Larsen et al., 2006), providing best practices, measurement, improvement and defining processes that belong to the five disciplines of service support and the five disciplines of service delivery (Sallé, 2004; Shahsavarani & Ji, 2011). The ITIL standard has been extended and adapted by IT organizations to provide their customers with a framework that is aligned to their own technologies. Four examples of these derived frameworks and their relationship to ITIL are presented in figure 12 (Sallé, 2004).

ITIL traditionally speaks of IT management *processes* in its first two versions. Up from ITIL v3 (or ITIL 2007), the naming convention changed to management *services*. ITIL v2 exists of nine books, of which two books, service support and service delivery, became the most used. ITIL v3 is reduced to five books (Service Strategy, Service Design, Service Transition, Service Operation, and Continual Service Improvement) and several complementary publications (Knapp, 2011). In 2011, ITIL v3 was updated and a new naming convention came into use. The revised version got the name "ITIL 2011" (Kempter, Kempter, & Lea-Cox, 2012).

Despite the reduction of ITIL books in v3, it still consists of many processes. Kempter et al. (2012) exhibit 37 processes and Knapp (2011) mentions a selection of 24, which can be seen as the 'core' processes. Since the two stages "Service Transition" and "Service Operation" cover the processes of operational management that are the subject of this thesis only the 'core' processes of these stages will be shortly described. The descriptions in this thesis emanate from the most recent version: ITIL 2011.

Service Transition

Service Asset and Configuration Management aims at maintaining information about Configuration Items (CI's) required to deliver an IT service, including relationships between CI's. Marten & Koenig (2011) state that services are constructed from service components and that service components consist of CI's that describe the internal structure of a service component. ITIL itself describes CI's as being any component of the IT Infrastructure, including documents related to the processes. However, the organization should define for itself which CI's need to be registered and which do not (e.g. because they are too small, inexpensive or irrelevant). The process owner is the configuration manager.

> *Change Management* is about control over the lifecycle of all changes that are made to the system. The primary objective is enabling beneficial changes while minimally disrupting IT services.

The subjects of *Release and Deployment Management* are planning, scheduling and control of new releases from the development environment to test and live environments. The rationale of this process is ensuring the integrity of the live environment and releasing only the correct components.

> *Knowledge Management* is more of a coordinating process aiming at gathering, analyzing, storing and sharing knowledge within an organization. Its primary goal is improving the efficiency of other processes by reducing the need to refigure knowledge that was already once generated (Kempter et al., 2012).

Service Operation

The process of *Event Management* is there to constantly monitor CI's and services. Its goal is to filter and categorize events for deciding on appropriate action whenever required.

> *Incident Management* manages the lifecycle of all incidents. An incident is an unplanned interruption of an IT service or a failure of a CI that has not yet had impact. The main purpose is to return the IT service to the users as quickly as possible.

Request Fulfillment is about fulfilling service requests of the users. In most cases this entails minor, standard changes or requests for information, e.g. changing a user's password.

> *Problem Management* manages the lifecycle of all problems. This contains a proactive part of analyzing incident records and data from other processes to identify trends or problems and a reactive part in which the process aims at minimizing the impact of incidents that cannot be prevented.

Access Management is about granting authorized users access to a service and preventing others from using it. The essence of this process is executing the policies set by the *Security Management* process, which is part of the "Service Design" stage (Kempter et al., 2012).

While the focus of ITIL is at both technical infrastructure and service management, ASL is a collection of best practices with the focus on managing application development and maintenance. It is the public domain standard for application management and complementary to ITIL (Larsen et al., 2006; Smalley & Meijer, 2006). Van der Pols & Backer (2006) describe that the ASL framework library consists of: generic descriptions of all of its processes, templates for documents that play a role in the processes, a standard set of agreements on metrics for reporting, checklists and an assessment for determining the maturity of the processes. All ASL processes are structured in a recognizable framework (figure 13), divided into six process clusters and three levels. The strategic level has a time focus of two years ahead and the management and operational levels have a short/medium term perspective. Of the six clusters, *Maintenance* and *Enhancement and renovation* represent the operational processes. These two clusters deal with the same application objects and are therefore closely related to each other. The two intermediate processes *Change management* and *Software control and distribution* control software releases and data enhancements from the development cycle to the live environment which is subject to the maintenance processes (Smalley & Meijer, 2006).

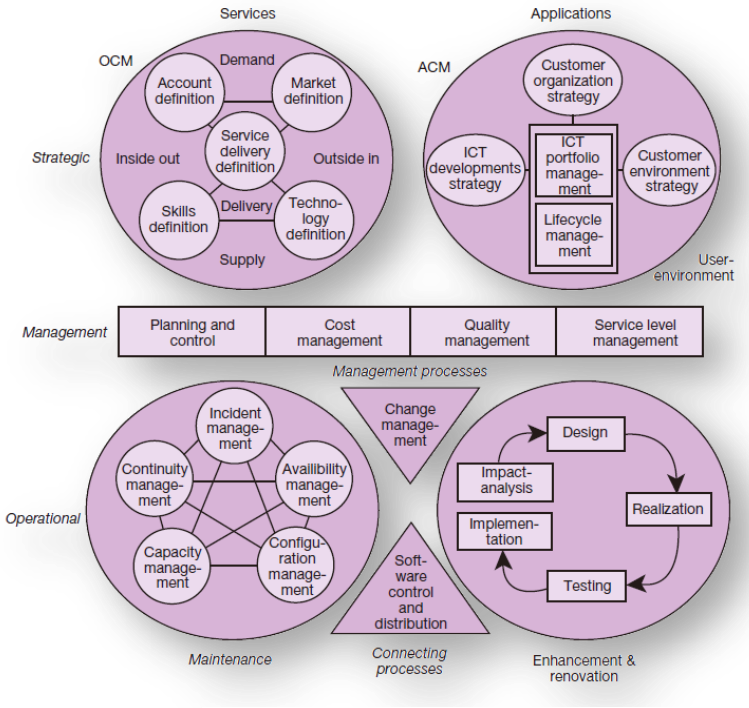


Figure 13: The complete ASL framework (van der Pols, 2008)

The five operational management and maintenance processes and the two connecting processes to the (iterative) development cycle will be briefly explained.

The function of *Incident management* is registering and handling reported defects and providing users with information when they have questions or when there are changes in the application or the management services.

Availability management controls all aspects concerning the availability and reliability of the application and management services.

Configuration management is about the identification, registration and management of the application and its management service components.

Capacity management manages the deployment of required computer capacity and assets to provide the appropriate management services and performance.

Continuity management is about ensuring the continuity of the service and the accompanying support. Examples that fall within the area of continuity management are security measures and back-up and fallback capabilities.

Change management determines which requests for change should be in a next 'release' of the software. The agreement on alterations to the software is the result of scheduling, costs and time considerations, in consultation with the customer.

Software control and distribution is about the management of software and associated objects during the phases of development, testing and transfer to the live environment (Smalley & Meijer, 2006).

2.3.3. CAPABILITY MATURITY MODELS

ASL can be complementary with a maturity model like Capability Maturity Model Integration (CMMI), since ASL provides a way to determine the maturity of its processes, but does not naturally use maturity levels itself. The focus of ASL is on how well the processes are implemented, while CMMI focuses on how well the processes are

managed (Smalley & Meijer, 2006). ITIL on the other hand does not necessarily provide the methodology needed to assess and improve the services by the use of assessments. To improve the service process capability, Niessink & Vliet (1998) propose the IT Service CMM which is closely related to Software CMM.

CMMI is the successor of the capability maturity model (CMM) and was developed with the aim to integrate many maturity models into one framework to improve the usability and improve the maturity over multiple disciplines. Before CMMI, many disciplines had their own separate Capability Maturity Model (CMM), e.g. for systems engineering, software engineering, software acquisition, workforce management and development and integrated product and process development (CMMI Product Team, 2002). CMM was traditionally developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University as from 1986 to help organizations improve their software process and to provide a method to assess the capability of software contractors (Paulk, Curtis, Chrissis, & Weber, 1993).

The goal of CMMI is to focus the organization on pursuing process improvement in multiple areas. This is being enabled by defining capability levels that include level-specific generic goals and generic practices. Capability level 0 is named “Incomplete”, meaning that the process is not or only partially performed. Level 1 is called “Performed” or “Initial”; it satisfies all the specific process goals. Level 2, “Managed”, requires aspects like a planning policy, skilled people, adequate resources and controlled outputs. The process is monitored, controlled and reviewed with reference to its process description. Level 3 is called “Defined” and prescribes that an organization has a standard ‘master’ process to provide as a basis for the defined process. The ‘defined process’ is like an instantiation of the standard process, improving the processes on both sides over time. Level 4 is named “Quantitatively managed” and involves control based on statistical and other quantitative techniques to evaluate the process based on quantitative objectives. The final level, level 5, is called “Optimizing” and appoints a process that is being changed and adapted to meet relevant current and projected business objectives. Important is continuous improvement of the process performance. All levels require their respective former level goals to be met (CMMI Product Team, 2002; Day & Lutteroth, 2011). The processes of ITSM can be matched to these maturity levels as been proposed by Niessink, Clerc, Tijdink, & van Vliet (2005).

Welke, Hirschheim, & Schwarz (2011) state that the maturity levels of CMMI refer to a type of orientation as opposed to the maturity within a fixed orientation. The authors propose a capability model specifically for SOA. As they explain, the initial level is about fine-grained software components, the managed level about emerged software architecture, the defined level focuses on business process support, the quantitatively managed level focuses on the higher level enterprise service architecture and the optimized level is about creating an adaptive architecture. As cloud services are also software services, organizations can also use these levels to indicate the orientation of their cloud efforts. Pereira & Mira da Silva (2010) and Pereira & Mira da Silva (2011) fit the ITIL v3 services into CMMI, showing which processes should be implemented into the organization to be compliant to a certain maturity stage.

2.3.4. THE CAPGEMINI MANAGEMENT FRAMEWORK

The Capgemini framework for application management, Way of Working (WoW), was originally created when the two companies *Capgemini OS Application Management* and *Capgemini BAS CAM* (former PinkRoccade) merged into *Capgemini Application Outsourcing* (AO), approximately two years ago. The first company particularly used ITIL and the latter was accustomed to use ASL. Since ASL is originally developed as an extension to ITIL, both frameworks complement each other well. Capgemini AO combined these two most commonly used industry frameworks and extended the result with CMMI and requirements for standards like ISAE3402, ISO27001 and ISO9001. The upper parts of the framework (OCM, ACM and service design) represent more of the ASL framework and the lower parts (service transition, operations and improvement) can clearly be found in ITIL. This does not mean that the other processes do not take place in the other framework, but the base is defined better by these frameworks. The global representation of the complete WoW framework can be found in appendix A.

The WoW framework must be seen as a means and not as a goal by itself, it has been developed pragmatically and is focused on the “How” (processes), not on the “What” (software development method). Therefore it provides procedures, templates, best practices, work instructions and training materials. It has gradually emerged in a continuous development cycle that is still going on. Since the framework is fundamentally based on frameworks that were developed according to best-practices in management of traditional software, it is not particularly composed to work for cloud services (Hofland & Molendijk, 2011).

To build a foundation for adapting such a framework to cloud services, the next paragraph describes the aspects that are distinguishing for this type of software.

2.3.5. MANAGEMENT IMPLICATIONS OF CLOUD COMPUTING

Gartner research (Dreyfuss, 2010) found that sound management of cloud services requires revision of the management practices that were developed for traditional outsourcing, since the delivery model is so different and the attributes are so unique. This means that research on management of IT outsourcing is not applicable to the cloud. These attributes do promise to increase the business value, but thereby challenge to manage the services in a totally new way. E.g. new support roles will emerge like cloud services strategist, broker, integrator, auditor and certifier. The challenge for service providers is to seamlessly integrate the framework of governance and management processes with the framework and processes of the client.

Since the emergence of SOA and cloud services, a distinction can be made between automated management tasks and the operational management tasks that need human interaction. Automated can e.g. be the provisioning of system images, monitoring performance and availability, metering and billing, monitoring and integration of applications (D. M. Smith, 2010). Rodero-Merino et al. (2010) state that some management tasks that are currently employed by humans can be automated. The authors mention e.g. automatic scalability and smart scaling to protect automatic scalability against malicious users. Another important aspect for many cloud using companies is the interoperability of clouds, in order to avoid lock-in as well as ease management functions. The Distributed Management Task Force (DMTF) is making an important effort on this subject with its Open Cloud Standards Incubator project. As published by DMTF Inc. (2009), their focus is on IaaS and management of the virtualized computing environment, addressing the aspects interoperability, portability and security. This includes SLA's, quality of service (QoS), workload portability, automated provisioning and

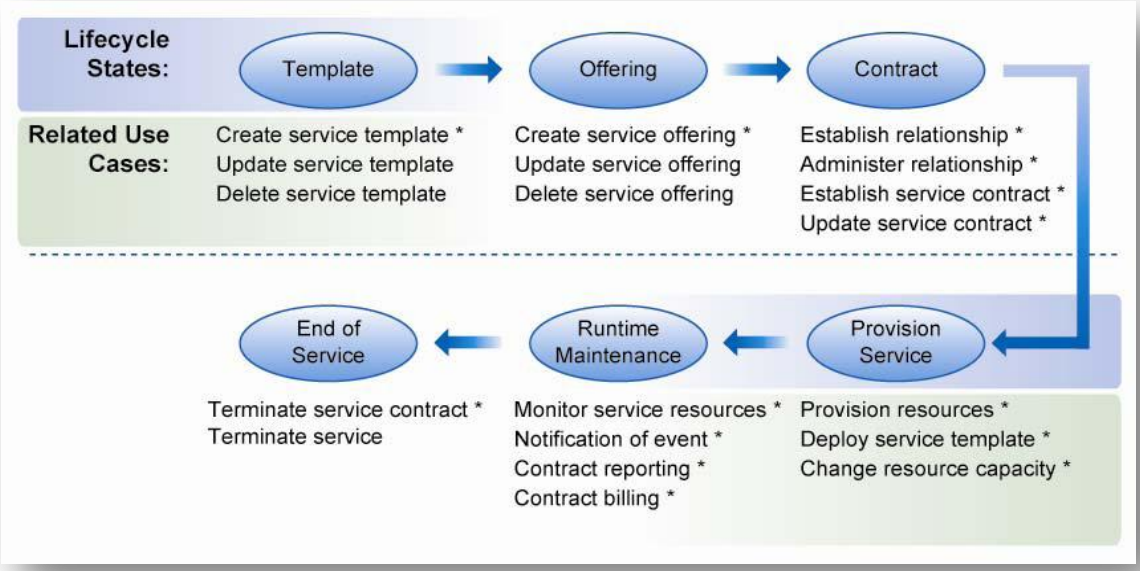


Figure 14: Cloud service lifecycle states and related use cases (DMTF Open Cloud Standards Incubator, 2010)

accounting and billing. The DMTF Systems Management Architecture for Server Hardware (SMASH) can automatically build and maintain a configuration database of both virtual and physical attributes, like virtual machines, configurations, assets and inventories. Another aspect of the standards for system virtualization management is consistent management of a virtual system's operational lifecycle; creation, modification, enabling, disabling, creating snapshots, and monitoring a system for these changes. The third aspect deals with monitoring and diagnostics of both virtualized and physical systems. Within the lifecycle of a cloud service, the DMTF Open Cloud Standards Incubator addresses the following management aspects: description of the service in a template, deployment of a service in the cloud, offering the service to customers, contracting the offering, provider operation and management of the service's instances, removal of the service offering (DMTF Open Cloud Standards Incubator, 2009).

Another standard in cloud management is Open Cloud Computing Interface (OCCI), also with the focus on integration, portability, interoperability and innovation, providing a protocol and Application Programming

Interface (API) to manage the resources (OCCI, 2011). The Storage Networking Industry Association (SNIA) created an ISO/ANSI accepted standard for data storage management and the Cloud Data Management Interface (CDMI™) follow-up standard for cloud data management. The CDMI standard supports CRUD activities as well as discovery of cloud storage offering devices, manages data and its containers and allows metadata to be associated with data and containers (SNIA, 2011).

Van der Pols (2008) argues that the adoption of a SOA approach will have implications for software management and maintenance. First, suppliers will also become customers of underlying services that are used to build a final service for the (end-)customer. Second, interfaces are needed to link services from several suppliers to the demand organization. Third, this means that integration becomes a dominant factor on both the supply and the demand side. Fourth, the focus will shift to the process output and making it fit. The actual process becomes less interesting. Fifth, the future strategy of the supplier becomes critical. Suppliers will have to compete with other service providers.

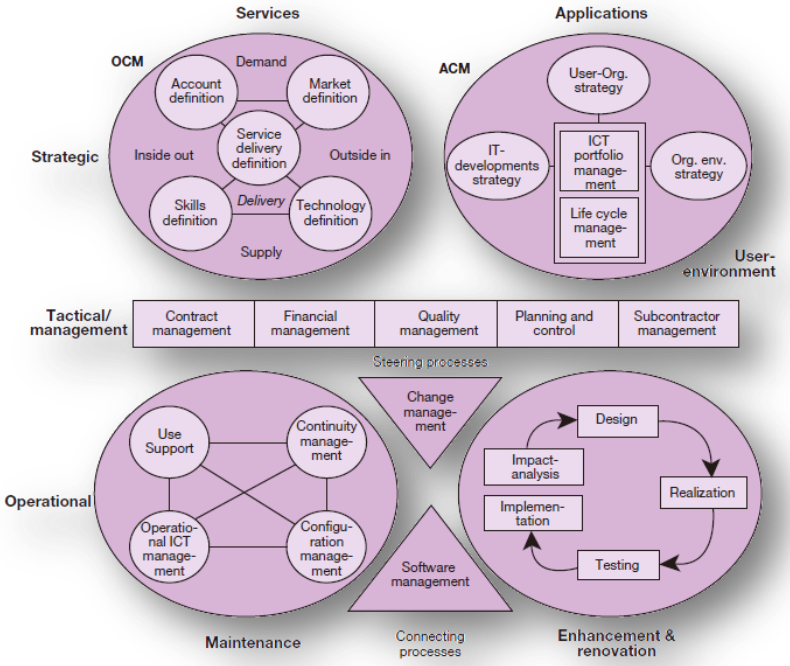


Figure 15: ASL fitted to SaaS, as proposed by van der Pols

The strategic process layer of ASL contains Applications Cycle Management (ACM) and Organization Cycle Management (OCM) which are concerned with the long term strategy of respectively the application and the IT organization that performs application management. These two clusters will not be affected when switching from traditional applications to cloud services. On the management and operational level, processes will need to be altered or are not applicable anymore, while new processes will actually start playing a role in the management model. A model proposition of ASL that adapts these changes can be seen in figure 15 (van der Pols, 2008).

Tardijn (2011) mentions some relevant aspects of changing management when switching to a cloud environment, first he states that technical management of the hardware-components and physical network will be performed by the infrastructure-provider. However, OS management is in most cases not included. This means that security maintenance and patches should be managed by the client company. As a second significant aspect is mentioned in this article that cloud servers always are distant, meaning that more management and maintenance has to be performed on the communication-infrastructure (security, performance and response times). The third relevant aspect is integration with the company's existing systems (e.g. with the active directory to be able to use single sign-on) and between services. The author concludes with the statement that service-integrators or cloud service brokers will have an important role. ADA Software (2012) describes the change management process when MDA is used. The two key differences with traditional development are that the business analyst actually makes the changes to the application business model and that underlying platform specific models and source code are automatically generated. Guo & Wang (2009) describe how the ITIL incident management process can be implemented to work optimally in a SaaS environment. The difference here is mainly in a direct collaboration with the SaaS provider to investigate and solve the incident. Next to some structural changes, the best practices of the operational management processes will be basically the same in a SaaS environment as in a traditional software environment. The importance of these best practices and process alignment to the organizational specifics has been illustrated by the case studies performed by Niessink & van Vliet (2000).

2.4. CHAPTER SUMMARY

CLOUD COMPUTING

Cloud computing is defined as the on-demand provisioning of resources and services. Within the term cloud computing, three general service models and four deployment models can be distinguished. The service model describes the level of the delivered service: SaaS (software), PaaS (platform) or IaaS (hardware). The deployment models indicate the deployment availability; public, private, hybrid or community. The advantages of cloud computing with respect to traditional on-premise applications are that resources can be dynamically scaled when necessary, that customers pay only for what is used and that there are no start-up costs. This lowers the barriers to try, test or start using a new application. The provider's main advantage is that hardware utilization can be maximized. Companies may, however, experience entrance barriers to start making use of cloud applications. Solutions are presented to overcome the ten main obstacles. Cloud computing is based on many SOA principles and the lifecycle phases are equal. The development method to create cloud services is often agile, utilizing the ubiquity of cloud computing to support the collaboration between developers, testers and live users.

SOFTWARE DEVELOPMENT

Conventional software is denoted 'native' or 'traditional' as opposed to web-based, cloud computing and SaaS. The first typically needs to be installed on a local machine and the latter can be used within any web browser. Software development has two main trends: waterfall and agile. The waterfall method is a process where all activities are performed once; e.g. testing only begins when the development is completed. An agile development method is characterized by the iterative and incremental building process; when the first part of the application is developed it will be tested and evaluated after which the next part will be developed, tested and evaluated and so on. In general there are three software delivery models: 'on-premise' where software is installed locally, 'hosted' where software is installed at another site and 'SaaS' where software runs in the cloud. MDA and MDD refer to a new way of developing software: visual programming, which can be seen as the 5^{th} generation programming language. A software application is created in a modeler and the source code will be generated by a model transformer tool which makes use of templates. Mendix B.V., a Dutch company, is a major party in providing a cloud environment that supports this kind of software development.

TRADITIONAL IT MANAGEMENT

(IT) Management is interwoven with many other terms, like governance, maintenance and control. IT governance can just as corporate governance be seen as the supervision of management-level processes, structures and relationships. IT Management is the operationalization which is shaped by the principles and goals of IT governance and it includes IT control and IT maintenance processes. Efforts in the past have standardized IT management processes based on best practices which have led to frameworks like ITIL and ASL, both compliant with directives provided by the ITSM method. ITIL is the worldwide de-facto standard in ITSM and ASL is a complementary and partially overlapping framework that provides a way to determine the maturity of the processes. It can therefore be used in combination with a capability maturity model. Capgemini IT and application managers have combined the formerly mentioned frameworks and ISO/ISAE standards to create their own framework of IT management processes. This framework and all underlying best practices are based on the management of traditional software. Cloud computing brings some fundamentally different characteristics that change the concept, technology and economics of software and makes traditional IT management frameworks less suitable. Some processes, mainly the ones affecting hardware, can be automated. ASL literature already mentions several aspects of the framework that may need to be changed to adopt the SOA and SaaS approach.

3. DEVELOPING THE SAAS MANAGEMENT FRAMEWORK

This chapter explains how the literature in the former chapter has been used to modify the Capgemini WoW framework in such a way that it should theoretically be aligned to SaaS deployment. The WoW framework as starting point has been modified according the ITIL v3 and the ASL SaaS proposition requirements, plus additional modifications that are needed to cover cloud aspects that are not accounted for in the former mentioned frameworks. After these adaptations, the first concept of the SaaS management framework arose (figure 16). The distinguishing characteristics of SaaS applications and the accompanying changes in the set of management processes will be explained shortly in the first paragraph. The second and third paragraphs elaborate on the changes within the change management and configuration management processes.

3.1. ADAPTING TO THE DISTINGUISING SAAS CHARACTERISTICS

Worldwide de-facto standard frameworks for IT management that are based on best-practices, like ITIL and ASL, are the results of years of experience and development. The information it contains can be seen as the outcome of lots of different case studies. For this reason, it is safe to assume that the processes these frameworks offer are effective and efficient and therefore these traditional frameworks will provide a solid basis for the SaaS management framework. The aspects of the ITIL v3 and ASL SaaS proposition frameworks that are adapted within the WoW framework will be described in this paragraph to come to a framework that fits SaaS solutions.

3.1.1. MODIFICATIONS BASED ON ITIL LITERATURE

ITIL version 3 does not only redefine the process-oriented naming convention to service-oriented, also some processes are changed to be better aligned to the special characteristics of cloud services. Jha (2012) explains the importance of these processes from a practical point of view and Cartlidge et al. (2007) explains the theoretical rationale. The following ITIL core processes are either completely new or differ on a global level in the framework for management of a cloud environment compared to the current management framework.

The processes that are changed in ITIL version 3 to be more in line with the needs of a cloud service environment mainly focus on the following aspects:

- The separation of management functions between provider and customer organizations.
- Service offering and service lifecycle phases
- Operational expenditure (pay-per-use) instead of upfront capital expenditure
- On-demand scalability of resources and services

SERVICE STRATEGY

Financial Management

Financial Management works in two ways: first, the customer organization is being charged based on its consumption and second, the organization's internal customers need to be billed. This makes that the process of financial management needs to be well defined and implemented. Details on resource utilization charged by the provider are needed to pass the correct costs to the internal cost center and for the provider to calculate the 'unit' of charging. The *Financial Management* process also contains budgeting, accounting and maintaining of the financial data.

Demand Management

Demand Management is mainly important from the service provider's point of view. This process should be directly linked to *Capacity Management* for the provider to be ready to deliver whenever the demand rises and to maintain a balance between utilized and excess capacity. The balance is important because of uncertainty in demand and excess capacity generates extra costs. *Demand Management* is there to understand and influence customer demand of services and accompanying capacity needs.

Service Portfolio Management

The service portfolio contains all services that are either proposed, in development, live or available for deployment or retired. The *Service Portfolio Management* process is about the management of the investment across the service lifecycle, supporting the process from studying market feasibility for a new service to deciding on its actual deployment and retiring.

SERVICE DESIGN

Service Level Management

The Service Level Agreement (SLA) will become even more important in cloud environments. The service provider operationalizes the SLA by Operational Level Agreements (OLA's) and Underpinning Contracts (UC's), which need to be very effective in order to meet the customer's requirements. *Service Level Management* negotiates, agrees and documents the service targets and then monitors to produce reports on delivery against the defined targets. An important asset of the process is that the performance of all services throughout the business is measured in a consistent manner. The coordination between *Service Level Management* and *Supplier Management* is important as well as an up-to-date service catalogue.

Service Catalogue Management

The service offerings and relevant service details are described in the service catalogue. This information is being used by the *Service Level Management* process as well as the business to view an accurate and consistent picture of the provider's services, including details, status and agreements. The service catalogue contains the available services from the service portfolio, namely the services with the live or available for deployment status.

Supplier Management

The *Supplier Management* process is fundamentally about building and keeping an excellent relationship with the service provider, so the organization can fall back on the supplier when needed. Its purpose therefore is to obtain value for money from suppliers and ensuring that suppliers perform as agreed in the contracts.

SERVICE TRANSITION

Service Asset & Configuration Management (SACM)

The *Service Asset & Configuration Management* process is renamed from *Configuration Management* and still entails the upkeep of the Configuration Management Database (CMDB) containing the IT components of the infrastructure and also of the services including shared assets. Important is not only identification, control and accounting for Configuration items (CI's) but also information on the relationships between them, since in a cloud environment one CI may be used to support different services provided to many customers. A failure of a single CI could in such an environment have a high impact on many users and thus on the business of the service provider.

Knowledge Management

This process is there to ensure that the right person has the right knowledge at the right time, to deliver and support the services the business needs. This results in more efficient services, improved quality, clear information on service's value and availability of relevant information.

Change Evaluation

Before developing or changing a service and deploying it, it is important to verify that the service is going to be useful and relevant to the business. Ensuring that the service will stay relevant during the Service Transition processes is being done by establishing proper metrics and measurement techniques.

SERVICE OPERATION

Incident Management

Incidents could in a cloud environment occur on two different levels: errors in the application/service and errors in the cloud infrastructure. Errors on the first mentioned level have to be resolved by the customer or managing organization and therefore needs proper (external) monitoring of events. The latter type of error is the responsibility of the infrastructure provider, however, it is important for the customer to make sure that the infrastructure agreements in the SLA are in line with their requirements.

Event Management

An event can be defined as a change of state that has significance for the management of a CI or service. It is important to monitor and record events and errors in the service. The difference between service or component monitoring and *Event Management* is that the latter generates and detects notifications, while the first checks the status, even if no events take place. The detection of an event may lead to an incident, problem, change or logging for later reference.

Request Fulfillment

The *Request Fulfillment* process is responsible for fulfilling service requests. The request of a cloud service can also be treated as a service request and does not need to go through the formal change approval cycle. Requests for expansion, modification or shrinkage of capacity or resources need to be applied in real-time, therefore no change approval cycle is used but the request needs to be equipped with the required approvals upon its submission.

Access Management

Access Management is there to provide access rights to users who are allowed access to a service or services and preventing access to users who are not allowed access. The process supports confidentiality, availability and integrity of data.

IT Operations Control

The *IT Operations Control* process includes routine operational tasks and centralized monitoring and control. The activities like job scheduling, backup and restore activities, print and output management and routine maintenance will typically be executed from a network operations center.

Service Lifecycle Management / Application Management

The *Service Lifecycle Management* process is involved in the application-related aspects of designing, testing, operating and improving the IT service. Also it plays a role in developing the required skills to operate the applications.

3.1.2. MODIFICATIONS BASED ON ASL LITERATURE

The modifications to the traditional ASL framework as proposed by van der Pols (2008) to fit SaaS software deployment are, like the ITIL v3 modifications, also based on practical experience. The ITIL framework has, however, gradually evolved through good practices, while the combination of ASL and SaaS is still just an idea of an author who is very notable on the subject. Remko van der Pols works as an architect and is chief editor of the architectural board for the ASL BiSL-foundation (The lifecycle company, 2012) and has written at least 19 books on the subject of ASL (Managementboek, 2012). The ideas of Remko van der Pols will be described in this paragraph and adopted as part of the SaaS framework to be tested and validated in practice.

Paragraph 2.3.4 explains that the Way of Working framework used by the Capgemini Application Outsourcing department is actually a combination of the ITIL and ASL framework, with additions by Capgemini. The ASL Framework (figure 13) and the insights of van der Pols on how to modify the framework to work for cloud software (figure 15) will be further specified here. The framework's processes that fall within the operational part and are affected by the change will be discussed one by one based on their respective section in the framework. The strategic processes (ACM and OCM sections in ASL) will remain unchanged, because for these processes it does not matter whether an application is (to be) deployed on-premise or as SaaS.

Recent literature on the ASL framework (van der Pols, 2008) describes aspects that were found as the result of remarks on ASL version 1.0 and of a fundamental strength and weaknesses analysis. The remarks take the following aspects of cloud computing into account:

- Suppliers also become customers of underlying services
- Input and output interfaces between multiple suppliers and the demand organization
- Integration of services from several suppliers; how, and who is responsible
- Processes become an internal affair; externally only the output matters (black-boxing)
- The supplier's strategy for the future
- Operational expenditure (pay-per-use) instead of upfront capital expenditure
- On-demand scalability of resources and services

Based on these characteristics, the book mentions changes in the framework. These changes will be shortly discussed, grouped by their process clusters. Appendix B shows the Capgemini WoW framework with these changes applied.

SERVICE DESIGN

Contract Management

The processes *Service Level Management* and *Supplier Management* have been merged into *Contract Management*, which has a wider focus including the following objects that are to be managed by this process:

- Responsibilities of the supply and the demand organization
- How and by whom the services are managed and funded
- The way of cooperation and communication between demand and supplier
- Assumptions and conditions that are prerequisite to ICT services
- Contract requirements and translating this to functional service levels

Subcontractor Management

This is a new process that needs to be introduced in an environment that makes use of cloud delivery. The idea of SOA and Cloud services is to reduce the complexity of a program by splitting it into smaller "black boxes". A service is likely to be built using the input from other services; this creates the need for making a right contract and managing the output of services that subcontractors deliver.

Financial Management

This process replaces the *Cost Management* process because the scope has been widened. The wider focus is necessary because of an increased use of cost calculation models. The models are used due to the translation of external charges to internal costs. This translation is needed in order to pass a fixed price and because of the increased use of subcontractors.

SERVICE TRANSITION

Software Management

The name *Software Management* may better fit the process *Software Control and Distribution* in a cloud environment since the distribution of software will be decreased very much. The only distribution takes place from development environment to test and live environments. More important becomes the management of software components and their respective versions.

SERVICE OPERATION

ICT Operations Management

The processes *Capacity Management* and *Availability Management* are to be combined into one process because of their similarities in flow and information and their impact on each other. The availability of a service is mostly influenced by its assigned capacity.

Use support

The process Incident Management should be renamed to *Use Support* in order to express a proactive attitude. Proactive communication towards users has always been important and is emphasized by this name.

3.1.3. MODIFICATIONS BASED ON THE OTHER LITERATURE

Based on the distinguishing characteristics of SaaS as described in paragraph 2.1, the former sections on ITIL and ASL literature do not cover all cloud aspects. The following characteristics also imply an effect on the management framework:

- Logical (customer) and physical (provider) infrastructure are different
- Instantiation: users work within their own application instance instead of the full application
- TCP/IP based; works over a networked medium (available wherever there is internet access)
- Creation, deployment and deletion tasks can be automated

Next to the characteristics that do affect the framework, there are also characteristics that do not need significant changes in the traditional framework. These distinguishing characteristics play a particular role in cloud service development and deployment, but do not influence the management framework in general:

- Software is service-oriented; abstract and accessible
- Strong fault tolerance; errors have less effect
- High security
- Data storage on a server (off-premise)

The implications of the aspects in the first enumeration are on two aspects: automation of certain management tasks and the difference between provider and customer tasks. The consequences to the management framework will be described here.

Automated management tasks

A lot of management tasks can be automated (paragraph 2.3.5). The literature mainly describes the automation of infrastructure tasks, which involve the execution of clear management policies. Once the policies are set, it can easily be performed by a computer. Automated tasks mainly apply to the infrastructure and may be used by the infrastructure supplier, although some processes should also be performed by the customer, e.g. monitoring, to verify that the supplier performs like agreed in the SLA. Monitoring on the demand (customer) side is included in the *IT Operations Control* process within the Service Operation category.

Application vs. infrastructure management

When customers work within their own virtual piece of hardware, the information technology overhead is considerably reduced, making cloud computing easier for the customer. The responsibilities between infrastructure supplier, application provider and customer should be clearly distinguished and covered by the agreements documented in the SLA('s).

3.2. THE CONCEPTUAL VERSION

In the Service Strategy category, the process *Application Portfolio Management* has been replaced by *Service Portfolio Management*. The process steps may need to be adjusted to the service lifecycle as opposed to the lifecycle of a traditional application. The ITIL v3 process *Demand Management* has been added. This process is mainly of importance to a service provider, but probably not unimportant to a customer organization.

In Service Design, the *Service Catalogue Management* process has been adopted. This process is essential to the service provider to express their offerings to (potential) customers. The *Financial Management* process is placed in the service design category because the billable 'units' should be defined and the viability to business units or employees should be considered. It is important to define these aspects in a stage before actually developing the service. *Contract Management* replaces the processes *Service Level Management* and *Supplier Management* in keeping the relationship with the service provider and ensuring that agreed service levels are being met. *Subcontractor Management* is a new process that deals with loose service components that are being provided by subcontractors as services itself.

In the Service Transition phase, *Configuration Management* will apply to different configuration items, depending on whether the process is deployed at a infrastructure & hardware provider or at a service provider. The first will use the process to manage hardware resources in the traditional way and the latter will use this process to keep track of service components. The *Change Evaluation* process, adopted from ITIL v3, focuses on verification, a focus which is also emphasized in the *Validation and Testing* process. *Knowledge Management* is added to support services more efficiently and provide clear and relevant information about services.

In the set of Service Operation processes, a clear distinction is introduced between application and infrastructure management processes. The first set is mostly performed by the service provider and/or the customer, the latter is mostly performed by the infrastructure and/or platform provider. Literature describes that many processes concerning infrastructure management can be automated and are not directly relevant to the higher-level service provider. The process *Incident Management* is renamed *Use Support* to sound more user-friendly and involves more pro-active communication to users. It can be combined with the *Request Fulfillment* process to be a single point of contact for users. *Access Management* organizes user access to the service and *Event Management* is there to generate and detect notifications. *Service Lifecycle Management* is an ongoing activity, directly involved in all the application lifecycle steps and therefore does not belong to one framework category, but most lifecycle stages are carried out in the operational phase and therefore it fits best in this category. *Operational ICT Management* is an aggregation of the former *Capacity Management* and *Availability Management* because these processes have much in common in a cloud environment and are therefore more efficient when combined. The process *IT Operations Control* is responsible for performing the routine operational tasks and monitoring of the service.

To clarify the different aspects of a model-driven SaaS environment on the general framework level as well as the process level, two of the framework's processes will be developed in detail. These processes are *Configuration Management* (from both the service provider's and customer's point of view) and *Change Management*. Both are processes with a significant importance in the Service Transition phase of the framework and they exist as well in the framework for traditional management as in the framework for SaaS management. These two processes kept their name, position and meaning in the global framework, indicating that they must be important. However, having no differences on framework level makes them interesting subjects for in-depth analysis and development.

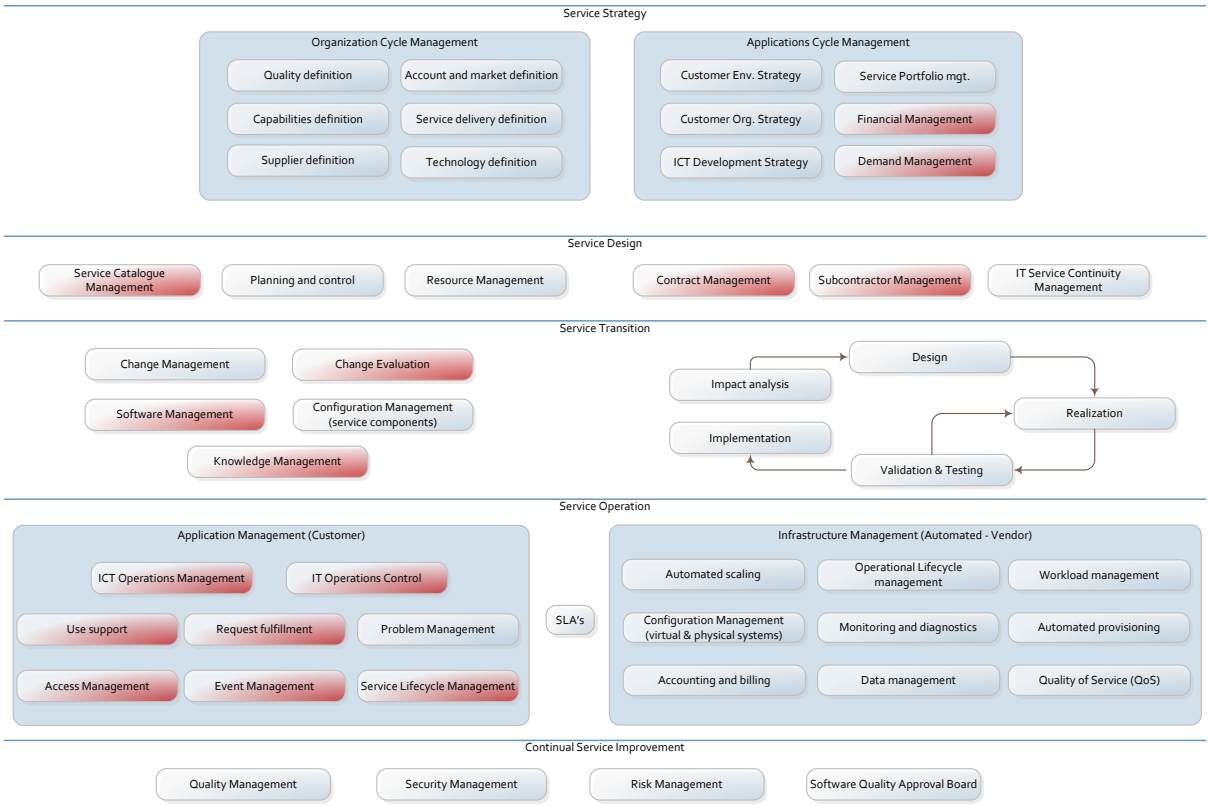


Figure 16: The SaaS management framework – 1st concept

3.3. CHANGE MANAGEMENT PROCESS IN DETAIL

The standard process model of the traditional ITSM/ITIL Change Management process is attached in appendix D. ITIL v2 and ASL describe that the change management process serves as a mechanism to identify, prioritize, initiate, evaluate and steer the desired changes, thus providing a standardized method to apply changes conform the organization's agreements (van Bon & van der Veen, 2007; van der Pols, 2001). The objective is to ensure a systematic process with the result that the system is in a well-defined state at all times (Mohan et al., 2008). The main activities in the non-model-driven change management process are as follows (in this order): Registration, Acceptation, Classification, Planning, Building and testing, Implementation, Evaluation. This process is closely linked to the Configuration Management process which processes the Change Management data and monitors the status of the Configuration Items (van Bon & van der Veen, 2007).

Traditional software development results in source code that can be maintained by software programmers. MDA gives business models which can be understood and maintained by business people, e.g. a business analyst. ADA Software (2012) states that when making use of model-driven engineering techniques, the main work can directly be done by the business units and IT will only be needed to transform the model to usable source code and deploy it on the hardware. Basically MDA will indeed eliminate the need to involve programmers in the development and change processes. However, this does require the modeling tool to support all possible programming features and the transformation tool to be able to generate all code automatically. At this time this is not possible, so programmers may still be necessary to manually create customized parts of an application to extend it with functionality that is not yet included in the model designer and transformation tools. When full MDA is supported, the planning, building and testing and implementation activities can be combined in the change management process since changes are modeled, tested and applied by a business analyst who needs much less time to do this, since it actually only means specifying the application structure and function in a model. The acceptation and classification activities can be merged with the registration process, since filtering, categorization and prioritization are superfluous when a change can quickly be applied and no other changes are pending. When no changes are pending and one business analyst can perform all kinds of changes, also the requirement of the planning activity becomes obsolete.

For the change management process it does not make much difference whether the application is being deployed as SaaS or on premise. However, SaaS may decrease the duration of the change process, since deployment to different development, testing and live servers becomes superfluous, making it possible to switch faster between these lifecycle phases. The creation of separate instances for each purpose will be sufficient and requires less time. Appendix D shows the change management process in a traditional environment next to a model of the proposed change management process (concept v1) in a model-driven environment. The fact that the environment is also cloud based does not alter the process, however, the model-driven aspect does.

3.4. CONFIGURATION MANAGEMENT PROCESS IN DETAIL

The standard process model of the traditional ITSM/ITIL Configuration Management process is attached in appendix E. ITIL v2 and ASL both describe that configuration management has two main purposes: keeping a reliable register of information on the organization's products and services and providing accurate information and documentation to support other business processes (van Bon & van der Veen, 2007; van der Pols, 2001). ITIL v3 slightly alters the objective on two aspects: it specifically mentions the importance of capturing information about relationships between Configuration Items (CI's) and it lays the focus on "CI's required to deliver an IT service" instead of managing "the organization's products and services" (Kempter et al., 2012).

Different definitions are used to describe the concept of Configuration Management (Dolstra, Bravenboer, & Visser, 2005; Krikhaar & Crnkovic, 2007). However, within IT management, the concept can generally be distinguished into two processes; *Software Configuration Management* (SCM) and the more self-evident *Configuration Management of hardware items*. The first has been a popular research topic (Krikhaar & Crnkovic, 2007). Both processes have in common that they keep a database about components and accompanying versions but companies used to focus more on the latter in earlier versions of ITIL. Version management as part of configuration management is especially applicable to SCM since information on bug fixes and new features or components may be relevant for supporting the software. Additionally, SCM encompasses configuration selection, build management, release management, installation management,

workspace management, concurrent engineering control and change management (Krikhaar & Crnkovic, 2007; Mohan et al., 2008).

Configuration management is a relevant part of the management framework for all parties that manage software and hardware components and their relationships. Deployment in a public or hybrid cloud environment implies that at least two providers need to keep a register of their own set of resources, creating the distinction between configuration management for hardware CI's and for service components as is described in paragraph 2.3.5 (management implications of cloud computing) and implemented in the first concept SaaS framework (figure 16). In the most straightforward situation, the IaaS provider only provides the resources and the SaaS provider provides and tracks the services. Whether the services are being built in a model-driven fashion will not influence the implementation of the configuration management process, since version and build management is mostly integrated in the development tools (Krikhaar & Crnkovic, 2007).

Appendix E shows the configuration management process in a traditional environment next to a model of the proposed configuration management process (concept v1) in a model-driven cloud environment. This concept v1 will serve as input to the case studies.

3.5. CHAPTER SUMMARY

SAAS IT MANAGEMENT FRAMEWORK

The Capgemini WoW framework served as the basis for the SaaS management framework, combined with ITIL v3 modifications, ASL SaaS proposition modifications and modifications from other literature on the distinguishing characteristics of SaaS deployment with respect to traditional software deployment. With ITIL version 3, the processes adopted or modified can be traced back to the separation of processes between service provider, broker and customer organizations, service offering and lifecycle phases, operational expenditure and on-demand scalability. The ASL SaaS proposition of van der Pols (2008) draws on the cloud computing aspects of underlying (subcontractor) services, interfacing between services, integration of services, internal/external relevance of the processes, the supplier's future strategy, operational expenditure and on-demand scalability. The distinguishing characteristics the former two frameworks did not account for are the difference between logical (virtual) and physical infrastructure, application instantiation, accessibility over a networked medium and automation of service creation, deployment and deletion tasks. These characteristics were covered by other literature to form the theory on how the framework should adapt them.

THE CHANGE MANAGEMENT PROCESS

The change management process is traditionally designed to provide a mechanism to identify, prioritize, initiate, evaluate and steer the desired changes. Its traditional main activities are: registration, acceptation, classification, planning, building and testing, implementation and evaluation. The process is close to the configuration management process which processes the change data and monitors the status of the items the change applies to.

In a full MDA environment, the acceptation and classification activities of the process can be merged with the registration process and the planning activity becomes obsolete because the business analyst himself can perform changes immediately. The deployment method does not change the process model but it may however increase its throughput time, since modeling, testing and implementation can be done on the same server (in different application instances if necessary).

THE CONFIGURATION MANAGEMENT PROCESS

The configuration management process is designed to keep a reliable register of information on the organization's products and services and to provide this information to support other business processes.

The implication of configuration management for a cloud service is that at least two organizations need to keep a separate register of their set of resources. This most likely creates a separation between the resources of the infrastructure provider and the service provider. MDA has no impact on the configuration management process.

4. TESTING THE FRAMEWORK

The framework concept in the former chapter is solely based on literature and practical applicability should therefore be evaluated. Even the modifications based on the widely accepted ITIL framework and experience-based ASL SaaS proposition cannot be accepted from literature alone, since the framework with its interactions between processes must be seen as one; it has been changed as a single entity. The framework phases, processes and interactions are part of the case study in this chapter and the expert reviews in chapter 5.

The first paragraph of this chapter explains the case study research approach. The data collection paragraph explains how the case studies have been conducted and preludes the actual case studies. The conclusions from the analyzed case data are described in the third and fourth paragraphs, elaborating on the companies, applications, interviews and interviewees. The cross-case analysis combines the output of the individual case studies, after which the theory modifications paragraph explains what changes should be done to the theory to make it fit practice.

4.1. CASE STUDY DESIGN

As stated in paragraph 1.4, this thesis consists of a theoretical and an empirical part. The first concept of the SaaS management framework and the configuration and change management processes has been based solely on literature. This theoretical concept of how the set of processes and processes should look like for model-driven SaaS applications will be tested by conducting a multiple case study; the empirical part of this research. The advantages of a multiple case study over a single case study are that multiple cases can produce more convincing evidence (Yin, 2003) and increase the research validity and generalizability. The goal of the multiple case study is to identify the applicability of the conceptual framework and processes in a practical environment. Important in the case study is the motivation of comments and practice, in order to improve the framework with it. On the framework level, the interest is not in the detailed processes, but in whether the processes are necessary and in which phase of the framework a process fits best. Regarding the two processes that were developed in detail, the main interest is in the necessity and effectiveness of its activities, checks, inputs and outputs.

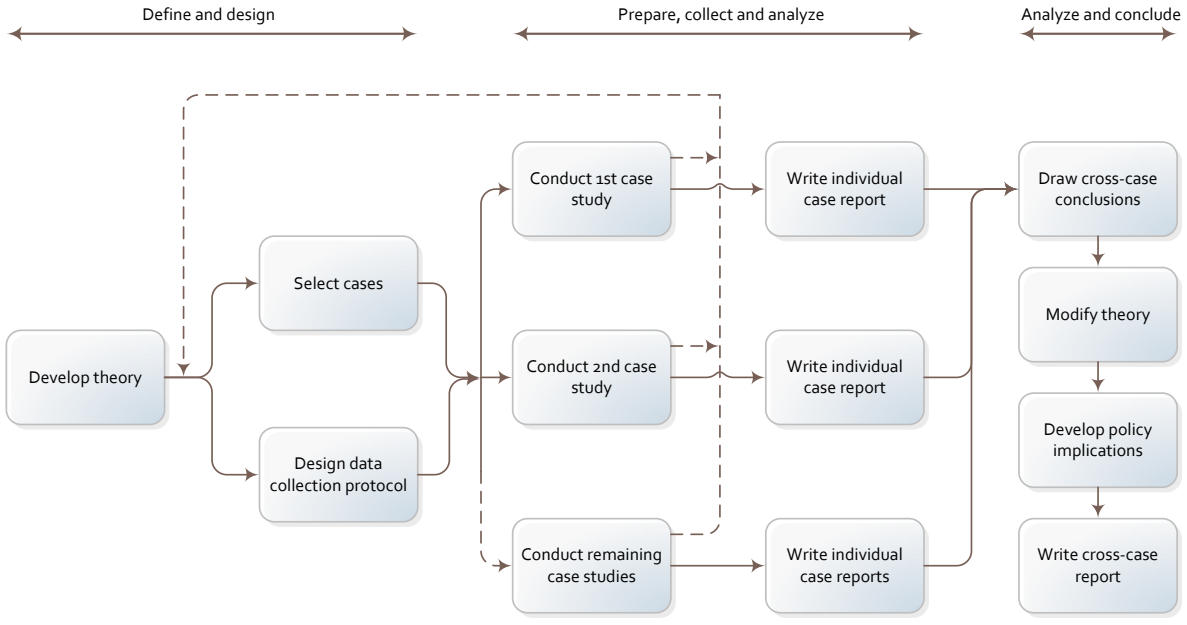


Figure 17: Case Study method (Yin, 2003)

The general flow of a multiple case study using the replication approach is modeled in figure 17. The replication logic enriches the theory that is being studied (Eisenhardt, 1989; Yin, 2003). After the initial activity of theory development, cases will be selected and the case study protocol will be designed. The selected cases should then be studied, all resulting in an individual case report. The dotted feedback line from the case studies to the design of the data protocol represents the situation that one of the cases appears not to suit the original case

study design. In such a situation, the design needs to be altered before proceeding with further cases. Cross-case conclusions can be drawn from the individual case conclusions, after which the theory will be changed to adapt the relevant discoveries and a report will be written on the definitive framework.

This model can be expanded with as many case studies as are considered relevant. However, the time factor will be taken into account. The more time passes between case studies, the more evolved applications and management may become in general, making it hard to make a fair comparison between cases and get conclusions from the older case studies that are still relevant. A time limit of four weeks has been defined in which all interviews should be conducted and reports should be typed and sent to the interviewees for approval. Because the research does not contain subgroups of extremely different cases and is well-defined, limited to both model-driven development and SaaS deployment on the Mendix platform, a high number of replications (more than two or three) is not necessary to be certain that the theory is effective in a practical environment (Yin, 2003). The aspects that can differ in this study are limited to organizational, e.g. company culture, size, industry, and to the application logic, e.g. size in function points, size in users and whether it is being used interdepartmentally. The expectation in such a limited environment is that 'theoretical saturation', meaning that adding more data has stopped enriching the theory because the observed phenomena has been seen before (Eisenhardt, 1989; Mohan et al., 2008; Strauss & Corbin, 1998), will be reached quickly. Eisenhardt (1989) describes that it is not uncommon to combine theoretical saturation with pragmatic considerations like time and money to plan the number of cases upfront.

4.2. DATA COLLECTION

Capgemini has only worked with the Mendix software since January 2012 and has already finished multiple projects successfully. This new way of developing software works out well, but another big part of services has not really been considered: software maintenance. The existing "Way of Working" framework is already categorized to the five phases of the service lifecycle according to ITIL version 3. These five phases are reflected by the five ITIL v3 core books. The phases are: Service Strategy, Service Design, Service Transition, Service Operation and Continual Service Improvement. Since the choice for SaaS deployment and model-driven development is a result of the strategic management processes, they should not change the strategic processes. Service Strategy is therefore disregarded as part of the case study. Continual Service Improvement is also excluded since this phase defines processes that should be considered for any kind of application in the generic level that is represented by the framework. Service Design, Transition and Operation represent the tactical and operational processes that are affected by the strategic choices of development, delivery and deployment methods and will therefore be part of the case study.

The literature review in chapter one shows amongst others: ideas to match existing frameworks to SaaS, operational management tasks that can be automated, and arguments to drop existing management processes and adopt new ones. These three subjects are the basis for the first concept of the SaaS framework that will be used to visualize ideas and provoke reactions from the interviewees. The focus points of the case study are:

- The distinction between vendor and customer processes
- The integration points between vendor and customer processes
- Automated management tasks
- The set of Service Design processes (mainly Service Catalogue, Contract and Subcontractor Management)
- The set of Service Transition processes (mainly Change Evaluation, Knowledge and Software Management and Validation & Testing)
- The set of Service Operation processes (Operational ICT, Event, Access and Service Lifecycle Management and Use Support)
- The Configuration Management process flow
- The Change Management process flow

The automated management tasks are mainly part of the cloud service lifecycle management as mentioned by Breiter & Behrendt (2009), which are: availability monitoring, service provisioning, storage management, metering and monitoring. These tasks are typically part of infrastructure management and should be performed by the infrastructure provider.

In correspondence with the research methodology, the first concept of the framework and processes is based on the literature in chapter two to provide a foundation for the case study. This step is important to form a vision on the matter, to be able to ask concrete questions to the interviewees and to base statements on visual models. After the propositions for the SaaS framework and processes have been derived from the literature, cases were selected, interviews prepared and stakeholders within the cases selected and approached to be interviewed. The case study focus points were discussed in open-ended interviews, i.e. by asking statements about the differences between the traditional and the proposed version and how the interviewee thinks this should work (Yin, 2003). The different aspects of the framework were assessed to find their relevance in practice and to ensure that no necessary processes have been overlooked. All aspects of the framework and process model first concept are changeable and up for discussion. The interviews were performed according to the case study protocol that has been made beforehand and is enclosed in appendix F. The case study protocol is there to ensure an equal approach of every case, increasing the reliability and validity of the research. After the case studies, a case study database has been set up and maintained to be able to browse and search it in a later stadium of the research. The chain of evidence between the collected data and conclusions in the report is maintained as much as possible to track all steps taken (Benbasat, Goldstein, & Mead, 1987; Yin, 2003). An interview report was created after each interview and sent back to the interviewee to approve and potentially expand with aspects the interviewee forgot to mention. To increase the case study validity, the quantitative technique of triangulation will be applied. More specifically; data triangulation, which requires the involvement of different sources of information (Guion, Diehl, & Mcdonald, 2011; Olsen, 2004; Tellis, 1997). The database is completely digitized, existing of the voice recordings of the interviews, all collected documentation and all reports. Relevant sources are employees who determine the management processes, execute them or are involved in them. Other relevant sources can be documentation in the form of process descriptions, input and output templates, memo's and observations at the case sites including amongst others the layout of workplaces, the way colleagues interact with each other and visual demonstrations of stakeholders performing actions on the platform that are related to the management processes. Interview reports, documentation and observations, if applicable, are listed in the appendices with the single case reports.

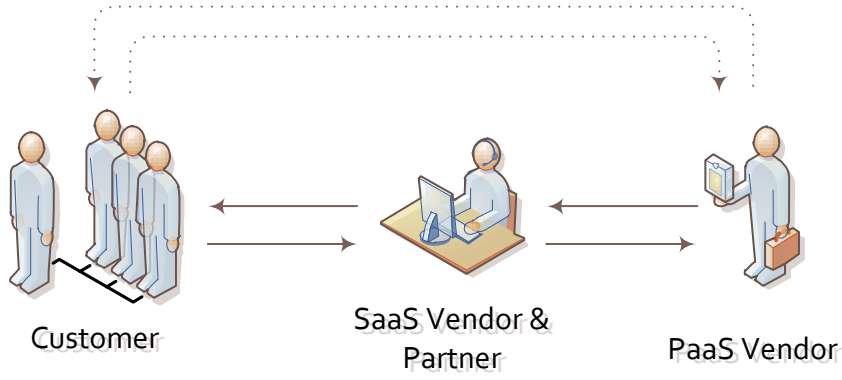


Figure 18: Relationship between customer and vendor or partner

In figure 18, the relationship between the customer and vendors is visualized. Mendix works with partner companies which develop services on the Mendix platform. Capgemini and Nobel are two of these partner companies. Once developed, both Mendix and the SaaS partner will aim at accommodating the application management with one of the partners. Not every customer company makes the choice to outsource application management to an external partner. When a customer makes the choice to manage the application themselves, like the PKN did, the management processes will be referred to as single-client management. Otherwise the management processes will be more adapted to multi-client management. Since the multi-client management model is considered standard and most desirable by PaaS and SaaS vendors, the lines between PaaS vendor and end-customer is dotted. These parties will not be directly related in this situation, although they may have formal agreements on the use of the platform.

When all selected cases are studied and the interview reports are approved, the collected data is used to improve the first concept of the framework and detailed processes. The changes led to the second concept framework, which was validated by expert reviews from PostNL and Capgemini (the AO division). Both organizations are experienced in developing and using the management processes, the latter mainly in

traditional environments but with some experience in SaaS environments (Tardijn, 2011), of which the Mendix environment is specifically emerging. The responses at this point of the research are used to determine and motivate to which extent the framework and processes can be used in practice. The necessary changes will be made and the result will be the third and final version of the framework, change management process and configuration management process.

4.3. CASE STUDY A – PKN

The case study starts with introducing the case organization and the application whose management has been studied. Next, the case study conclusions and additional findings are presented. The interview reports as well as the documentation used to conclude on this case study can be found in Appendix G.

CASE INTRODUCTION

The organization

The "Protestantse Kerk in Nederland" (PKN) is translated into "Protestant Church in the Netherlands". The organization originates from the merging of three protestant denominations in the Netherlands: the Netherlands Reformed Church, the Reformed Churches in the Netherlands and the Evangelical Lutheran Church in the Kingdom of the Netherlands. The PKN has existed as one organization since the first of May 2004 and currently has around 2 million members spread over 2.000 local congregations with a centralized service organization in Utrecht.

The application

The application "Leden Registratie Protestantse Kerk" (LRP), translated "Member Registration Protestant Church" is the application that is used to keep the member data of all 2 million church members and related persons. The application consists of four major modules: organization and structure, members, fund raising and support of pastoral and church work. There are four user roles: control, members, funds, pastoral. The potential turnover on fund raising is approximately € 400 million. The application is web based and developed on the Mendix platform. The initial claim was that the application should be able to be used by 10.000 concurrent users. It has been scaled and proven up to 2.000 concurrent users. The current amount of end-users is somewhere between 15.000 and 20.000. LRP is the largest application built on the Mendix framework so far, both in terms of resources and users. The building of the application with multiple iterations started in January 2010 and de application was commissioned a year later in January 2011. The following actors have a role in the management processes of LRP within the central service organization of the PKN: 6-8 people service desk, ± 3 people order handling, 2 people application management + one Mendix specialist for 0.4 FTEs and the team leader of the LRP team. All are under the leadership of the head of the ICT department. The formal owner and budget holder of the application is the head of the institutional support department. Furthermore there is a user platform that directs user wishes.

The transformation project from the former distributed applications to a single application included the conversion and migration of more than 800 local databases. The LRP project followed up on another project that attempted to adjust the standard Oracle HR software package. This project was called "Numeri", cost € 5 million and failed after 4 years. The second project, LRP, eventually cost € 3,5 million and is expected to save € 861.000 on a yearly basis, mainly because fewer people are needed in the central organization since local congregations can manage more data by themselves. The project followed an agile approach with iterations of 1-2 weeks. The project team existed of a product owner, project manager, concept controller, 2-3 Mendix builders and included 50 - 100 voluntary project members (Kieskamp, 2011; Sanders, 2009).

CASE STUDY CONCLUSIONS

The SaaS management framework

- Service catalogue management is not really applicable because only one SaaS application is deployed from Mendix: LRP. However, Mendix does offer additional services to this application, like the fallback environment and management of the hardware. Mendix has recently developed a tool for service catalogue management that will be released in a few weeks. This tool is developed to keep track of

the “app-jungle” and to give insight in versions and the deployment status of an application, including who has access and who is working on it at the moment.

- Contract/subcontractor management: from the PKN’s point of view Mendix is the contractor for the hardware management and XS2OS for building projects. A subcontractor from Mendix is XS4ALL, since they provide Mendix with a physical location, internet line, hardware rack, UPS, etc. for the servers. The IaaS is provided by Linode. LRP is a special case among Mendix applications, because the PKN does not work with a partner company for application management but perform it themselves.
- Change evaluation: it is important to have a clear understanding of what the user is really requesting. Only when his/her wish is clear, it can be evaluated to decide whether it would be a good change for the system as a whole. The metrics on the change are then already set; they are the aspects that are part of the initial request. Change evaluation mainly takes place at the beginning of the change process and during the iteration. Not after a change.
- Software management is not specifically arranged for LRP. The Mendix modeler tool keeps track of different versions of the model and the only distribution is from the test to the production environment. Mendix does not (yet) work with software components, so there is no need to manage versions of application components. The advantage of a SaaS application is the absence of the need for software distribution management. You are always sure that everybody uses the same version of the application and your users need not to fiddle with installations. The disadvantage is that everything will be out when you do something wrong on the central system. Another disadvantage is the disparity between browsers. Especially older browsers can cause unexpected situations. A new version of the application is released around dinnertime or early in the morning. These are the time periods that the end-users make the least use of the system, since the majority of them work voluntarily.
- Knowledge management is important to provide LRP support and application management. Multiple databases with knowledge are maintained that are used by different groups of people (end-users and service desk/application management). The most important tool to the latter group is ‘Sprintr’ to keep track of any problems and their solutions, even for simple questions.
- Access management makes use of delegation of control. The service organization staff has many authorizations. Then, every local congregation has a local manager who has the authorization to give people in the congregation access to a compartmented part of the system. To be able to access the application, you will need three things: a valid username, password and a generated token. Access to the actual server files (e.g. for application managers) is regulated by Mendix.
- Event management is on the first hand a system functionality in LRP. A type of “event-driven marketing” is used to notify the system’s users if an event has occurred that is specifically relevant to that user. This manifests itself by an overview window. On the other hand, there is event triggering on e.g. the processing of “Stichting Interkerkelijke Ledenadministratie” (SILA) mutations. Also, when the system goes down it will automatically restart.
- Service lifecycle management is split in two ways: regular maintenance (patches) and the build of new versions on project basis. The regular maintenance skips the “design” activity since it aims at the repair of existing functionality, not building new functionality. The build of new functionality is realized by partner XS2OS and based on the formalized description the PKN prepares on the functionality that should be built.
- Use support: 1st line is the telephonic service desk. This is the first filter for questions and issues from end-users (external). The filtered feedback from the helpdesk as well as feedback from service organization users (internal) goes to the 2nd line application managers who test, filter and prioritize it. The application managers repair bugs in order of priority. For extending current functionality, building new functionality and optimizing functionality the partner XS2OS is hired on project basis.
- Request fulfillment also starts with a user wish coming in at the service desk. This process keeps track of the list of wishes, the GPL grades them and once or twice a year new functionality is realized from this wishlist.
- Problem management is not a distinct process. It is recognized by the service desk and/or application managers when an issue comes in repeatedly, after which its priority will be increased because of the structural nature. Also issues that completely block a local congregation or an important centralized process get a high priority. The service desk and application managers decide on the priority and therefore on how much an issue/incident is a problem.
- ICT operations management and ICT operations control are performed by Mendix. They monitor and check the hardware and let us know when a new piece of hardware is required to extend capacity or

fix a broken piece. Also making backups and job scheduling is something Mendix does. Availability of the application is covered by the SLA with Mendix. The PKN application managers only make a backup themselves to restore onto the test environment.

- Monitoring tools from Mendix are in place to watch the platform. Mendix and the PKN can both view the output from these monitoring tools. When the system goes down or reaches a critical state, the tool automatically sends an alert to the LRP application manager by SMS or e-mail.
  Another automated process is the execution of analysis and correction scripts every night, e.g. to repair specific data.

To create reports, specific analyses can be manually run by the application managers. The application also has a 'statistics module' that gathers and assembles statistics every month.

When working with a provider for the hardware and the platform, it is important that they can prove and demonstrate that they are in control of their management processes. Many things do not require your attention when you outsource them, but you should not take them for granted.

The framework does not lack any process or processes that are performed in the context of management of LRP.

Change management

For a user or the GPL to submit a change, a structured description is required in the form of a template including a rough draft if possible. This clarifies the wish for a change, making it easier for the internal application managers or external developers to build it. When it is initially clear, it will also take fewer iterations before the user can approve the change. Furthermore, input for the process comes from legislation, church order, requirements from 3rd parties (like European banks with SEPA) and internal departments. The service desk is the first filtering mechanism after which it has been registered, then the application managers assess and prioritize it. Categorization and prioritization are important when there is a lot of work to do. There are no special urgent procedures other than setting a higher priority. A special planning activity would be redundant because the priority dictates the order. Then, a sequence is used of local development (= model) -> deployment to test environment -> test -> (correct, means back to local development) -> accept -> deploy to production. The transformation is done by the system in the background. Status monitoring by configuration management is not applicable. Since the change has been thoroughly evaluated beforehand, an evaluation activity is not necessary after the deployment.

Good testing is important in the change management process, since the use of a pilot group in the production environment is impossible. Communication towards the end-users about changes is also important.

Configuration management

The only configurations relevant to the PKN are the software on the test and production servers and the browser versions. A distinct process to take care of this is not needed. The application does not exist of service components that require being tracked. The application is seen as a whole. The expectation is that the future version 5 of the Mendix platform will offer the ability to work with different, connectable modules.

Mendix takes care of the management of the hardware, OS and the Mendix platform. The PKN has made clear agreements with them about server management. Mendix manages both the production environment and fallback environment for them. Hardware-level (IaaS/PaaS) configuration management is therefore performed, but software-level (SaaS) configuration management is not necessary. However, keeping the hardware tuned to the application requirements at all times is an important part of hardware configuration management.

Additional effects of model-driven development

The Mendix framework does not support the creation of application documentation. The model is actually its own documentation. However, in specific situations the original model developer should have had the discipline to describe (e.g. in a comment) why he has made a specific choice. An advantage of non-agile development methods is that the system design documents can be used in practice as the system's documentation after its development.

Additional effects of SaaS deployment

The choice for a central, web based system fits in the goal of the church to unburden the local congregations in the coming years. An important aspect of a large centralized application is to break it down in smaller units of

user accessibility, so someone with access to the system data can only view and modify the specific data that are relevant to his geographical and functional responsibility. An advantage is that the local congregation does not need to perform application management, however this makes it also impossible for them to make a preventive backup and play around in the system. Local congregations cannot be individually backed up and restored, since data from one member can be used by more than one congregation. To give the local congregations an environment to play around in, a test environment is provided which is being refreshed monthly with actual data.

An issue with SaaS deployment is that it is difficult to use a certain group of actual end-users as pilot testers before implementing a change to everyone. Once a change has been made to the production environment, it will be there for every user. The options you have are limited to the two environments of test and production. Testing can only be done in the test environment with test data. It is not able to synchronize the edits users make here with the production environment.

ADDITIONAL FINDINGS

The easiness by which additional functionality can be developed with the Mendix framework is of course an advantage, but it also has a downside. The tendency is to think that you could pick up all kinds of requests while you are working on some functionality. This should be avoided, because it gets quickly out of control (and time and budget). For the development of LRP, the users were therefore made very clear that they should tell which functionality they *need* to have and not what they *would like* to have.

Another one of the PKN's good practices is displaying the application version number in the right top corner. Because documentation is kept on which version solves a specific issue or feature, the helpdesk can look back whether a user problem has already been fixed.

Issues and requests should be submitted well described in a standard template. This relieves the application managers and shortens the agile development process, because less iterations of development and testing are necessary before the developers have built exactly what the user wants. The requester should be included in the iteration's testing activity to make sure that it was developed the way he meant.

The Mendix platform does not support read-only or "observer" authorizations. Such authorizations would be convenient to show development progress to people who are not allowed to make changes or e.g. the financial chief who accounts for it.

4.4. CASE STUDY B – NOBEL

The case study starts with introducing the case organization and the application whose management has been studied. Next, the case study conclusions and additional findings are presented. The interview reports as well as the documentation used to conclude on this case study can be found in Appendix H.

CASE INTRODUCTION

The organization

Nobel is a strategic ICT partner for the mid-market, offering both cloud platform and on-premise services to support the business of their customers. Nobel exists of three business units: Managed ICT, Business Solutions and Document Group. The business unit Business Solutions aims amongst others at SaaS concepts (Persberichten.com, 2011). The company employs around 350 people and is one of Mendix's channel partners since 2010. Other Mendix partners are Capgemini, FICO, Capegroep, Mansystems, Flowfabric, Atos and Centric. These companies offer the Mendix platform to customers, develop applications on it and perform the management. Since Nobel performs these tasks for multiple customers, they make use of generic multi-client management processes. Nobel Director Piet van Vugt motivates the partnership with Mendix as follows: "*We see that our clients need the ability to quickly modify their strategy and business models in order to adapt to the rapidly changing market conditions. This means that businesses must continually adapt and optimize their processes in line with ever changing business strategies. The traditional ERP software provides insufficient flexibility to quickly and easily implement these changes*" (Kepinski, 2012; Roos, 2010).

The application

Nobel performs the development and management of multiple applications, this case study will therefore not focus on a single application but on the management processes that are in place for all applications in the same way. Nobel is besides Mendix partner also partner of Exact, and their management experience with different hosted and traditional applications enables comparison between traditional management and model-driven SaaS management. The added value of specifically a Mendix application replacing a traditional application is therefore not applicable in this case. The management processes Nobel uses are based on the conventional ITIL processes.

Almost all Mendix applications that are managed by Nobel are in their own cloud environment. Since the applications are not in the Mendixcloud, not Mendix but Nobel is responsible for the hardware and infrastructure. All Mendix does is providing the platform and providing updates to the platform as stated in their SLA. Nobel manages about 15 Mendix applications on 5 or 6 large environments. All applications are built by themselves and they are currently developing two new environments. One of Nobel's applications built on the Mendix platform is an EDI application for the interchange of electronic messages. Other applications support business processes. The Mendix applications managed by Nobel vary in size from 5 to 100 end-users from within the customer organization. The early applications are used by SME's and nowadays Mendix development is focused more on SME+ and corporate size. The actors in the management processes are: the team leader service desk, the service desk employee, the coordinating service manager and the Mendix consultant and his manager. When escalating, the manager services business solutions gets involved and when it escalates even more, his manager is involved. When it is even larger, it could be picked up company-wide. The service desk consists of five persons who are responsible for the workflow of open incidents and the dispatch to the right team within 14 product lines.

CASE STUDY CONCLUSIONS

The SaaS management framework

- The difference in financial management is that the customer buys a service with a flexible license that can be terminated monthly. This means that the contract is different, it is called an SPLA (Services Provider License Agreement). In Nobel's auto-provisioning environment, the customer is billed based on the amount of users he adds and the applications they use. Mendix makes use of less flexible user licenses. The financial management process is associated with contract management, subcontractor management and IT service continuity management.
- Demand management is part of technical management. At a strategic level, this process should be about discussing with the customer how the data should be dealt with. It is important to define a threshold on the capacity and to discuss with the customer what to do when the threshold is reached to avoid problems with the continuity of the service.
- Service catalogue management is a confusing name since Nobel makes use of a Products and Services Catalogue (PSC). Traditionally products would refer to an application and services to the management services. The PSC describes what Nobel offers (both the applications and services), what the customer can expect and what is involved. Mendix does not offer a service catalogue, they only offer the development platform.
- Requirements management could be a good process at this level when the software delivery does not think from the direction of which standard applications there are to offer, but from what the customer wants. The Mendix platform is typically used to create custom programs specifically aligned to a single customer, so requirement analysis is an important aspect.
- Contract management is about the SLA with the customer. This may be combined with subcontractor management.
- Subcontractor management could be an agreement with Mendix when Nobel would host on the Mendix cloud environment but also the maintenance/delivery terms of the Mendix platform. The latter is a standard agreement Mendix uses. Subcontractor management is also applied when Nobel uses the Mendix platform to fulfill a large part of the customer's requirements and a linked software package from a third party supplier to complement it (e.g. a content management system to update data).
- Service level management should not be replaced by contract management. The interviewee believes that the latter should be more about watching over the contract and service level management entails

a lot more. The process is responsible for additional revenues, being physically present at the customer site and see/hear things that do not work well, lead escalations and plan consultancy. Visual presence is important at the customer's site. Contract management is more internally focused and the service manager external; to suppliers and customers.

- Change evaluation does not add much value when the change management process defines a CAB (Change Advisory Board) and there is a release management process in place. Keeping the conclusions of CAB considerations is difficult in practice. Evaluation is always a good thing, but it is usually skipped in practice. However, if there is an evaluation, those results are not used.
- Software management is strongly related to configuration management. At application level, distribution may still be necessary in a cloud environment, since the application may be hosted at multiple sites. System management shifts from client to supplier. Important in a multi-tenant environment is to check whether all instances still work after an update of the environment.
- Disaster recovery should be discussed at strategic level: storing the data at multiple locations will be more expensive. Recovery times and how data are synchronized should be discussed.
- Knowledge management involves the use of a knowledge database that contains the solutions for problems and known issues. The service desk could find a solution here after their analysis of an incident. For the "Exact" software the incidents and solutions are imported in some KM system.
- Access management is an aspect that many customers like to perform themselves (granting access, setting rights and roles). The customer's application manager performs this task. He will have extensive access rights. This is identity management. License management is related to this process because sometimes the license needs to be extended before more users can be added.
- Event management can use the warning that the storage threshold is reached as input. Service- or Account Management will then contact the customer to discuss whether capacity should be added or current storage may be cleaned up.
- Service lifecycle management may fit better in the continual service improvement phase because it entails all service phases. A service manager could coordinate service development, testing and deployment from this process. The process will be linked to change management for the part of improving the service. The process offers the opportunity to pro-actively improve the customer's service e.g. based upon market developments.
- The use support/incident intake process consists of a 1st line service desk and 2nd line Mendix consultants (who are the modelers). A service manager keeps track of the incident processing. When a consultant is needed, the service manager will make sure this consultant is available for solving the incident. This process could be more efficient when the 1st and 2nd line service desk are integrated, or when more of the 2nd line tasks can be performed by the 1st line. If everyone at the 1st line service desk is trained in "process thinking" as used by Mendix, the 2nd line would only be needed when custom scripts with for example Java-code are involved. The incident process is the most important process. Pro-active search of issues is important for solving customer issues before the customer notices them.
- Request fulfillment should actually be a separate process from incident management, because if it is part of incident management for example requests for information may be overshadowed and get a low priority. Requests can come in through the service desk, service manager, administrator, engineer or a consultant.
- When a customer buys the problem management service, a problem manager will be appointed to analyze and solve recurring incidents. The information about these problems is then used for one or multiple customers. Problem management has not only quantitative KPI's, like incident management, but mainly qualitative KPI's. The fundament of Problem Management is proper analysis to prevent incidents in the future.
- ICT operations management can combine availability management well with capacity management. An example of practical use of the latter is placing a threshold at a certain percentage of the storage capacity creating an event when this is reached.
- ICT operations control entails scheduled jobs and backups. The latter is important, also in a SaaS environment. Within Mendix, former releases of the model should be backed up to revert to if necessary. This may be included in the platform software.
- Automation can be found in the change, problem and incident management processes based on their workflow within the service management system. Monitoring of hardware and network are automated. There is no monitoring at the process level. It is always an issue to get applications work perfectly together. An important automated aspect is fault management; when a fault is detected you

can act on it with e.g. a certain management task or a visit of the account manager to the customer. Mendix also has some monitoring function.

Preconditions for management are good agreements with the customer (expectations management), which is covered by the SLA. Also a DTAP street for development, testing and acceptation before deploying it is important. In the case of connecting with an application from a third party supplier, supplier management/third party management can be part of the SLA. Nobel will then figure out where the problem lies when the connection does not work and take care of the communication until a solution is provided.

Not only SLA's, but also UC's (Underpinning Contract) are part of the operational processes. The SLA describes the quality of the services while an UC deals with underlying parties.

SLR's (Service Level Reports) are periodically generated for the Service Management customers. This is an important report because it shows the customer's changes, incidents and how the incidents were solved. Next to the SLR there are some reports for internal use.

Change management

Besides the already included sources, RfC's (Request for Change) can also be registered at the service desk through service managers or from the event management process. The input, however, will mainly come from incident and problem management.

An important step in change management is customer approval, since a change can involve costs.

There is always someone who should make the change, so planning is still important. Planning is especially applicable because people are also working on other things (both development and support). Development of changes in Mendix will go faster.

Categorization and priorization is important to identify whether a change should be implemented more rapidly. If that is the case, the urgent procedure is invoked.

Change management should have a link to release management, since a new release should be issued after the changes have been made.

Some judicious body should be involved in the change management process to decide whether a change could conflict with the current functionality.

The impact and resources activity should contain an analysis whether the change has consequences for other parts of the system.

When the change has been developed and tested, you can still find out that an error has been made in the classification. The process should therefore offer the option to go back to the classification activity. The right order is: build, test, see if it works, implement.

The evaluation activity could be used to modify the process if the arrow "no" has been followed at the decision point "does it work?".

The change management process is performed by the Mendix consultants. Since an application is stage-wise developed in a project, consultants are continuously working on it and can easily make the required changes. Most of the times an analysis will be done, but there is no RfC like in the conventional process. For Mendix development, a DTAP (Development, Testing, Acceptance and Production) street is used.

Configuration management

Regarding software configuration management, Nobel registers which application the customer uses and the accompanying version number. This should be updated by the change management process after a change has been made. With regard to Mendix, it is not necessary to register the version number, knowing that it is Mendix is sufficient.

Currently, there is no need for defining naming conventions. Registering and updating CI's from the change process should be performed consequently. The monitoring and auditing activities are not performed on the CMDB.

Configuration management especially takes time to keep. When a lot of characteristics need to be remembered it can be very convenient. However, when tracking a few characteristics of a few applications, constructing a database does not have priority above other processes. Employees can remember it themselves.

ADDITIONAL FINDINGS

When developing a new framework, use of conventional terms may be a good idea, since the users of the framework are accustomed to them and therefore will understand the framework quicker. For example the difference between Incident Management and Use Support is only the name.

The traditional incident management process could be used for “agile management”. However, the delivered functionality of the application should be documented clearly enough before it can be transmitted from development to management. Uniformity of development is an interesting aspect to understand the developed models better.

It might be a real option in the future to have standardized templates of common processes that are modeled in Mendix. These templates could be based on best practices and with some modifications for the specific customer deployed for that customer. This will save development time and bugs in the model (since the standardized model is used before, many bugs will already have been eliminated).

4.5. CROSS-CASE ANALYSIS

The cross-case analysis shows the aggregated conclusions from both case studies. Both the similarities and the differences that have been found when comparing the case studies will be described briefly to establish the necessary modifications to the theory.

SIMILARITIES

- Mendix does not provide service catalogue management, but they are working on a tool to support it in the future.
- Contract management is generally about the agreements between customer and provider. It includes service level management, so it works at both the customer as the provider site.
- Subcontractor management is in order when the provider outsources part of the service he delivers. An established term for this is “underpinning contract”.
- Change management is still an important process. Input can come from multiple parties. The service desk is the first filtering mechanism and then the change is prioritized. There may be special procedures for urgent changes when multiple applications are managed. This has to do with the planning. The same applies to a separate planning activity. In the case of multi-client management, a customer approval activity should be taken care of. Also a judicious body should be involved to evaluate the change. The transformation happens in the background and an evaluation would add value in theory but is not used in practice.
- Configuration management is not really applicable on software level. Mendix does not support the use of different modules. At application level, knowing which application is deployed is sufficient.
- Knowledge management is important to connect workarounds and solutions to known errors. It is also convenient to describe how the application works.
- Access management should be performed close to where the system is being used. A delegation principle to the customer or local user groups is used in both cases.
- Event management comes into action when an event is triggered. This may be the result of monitoring or being integrated in the system.
- Service lifecycle management entails all service phases. The process has a coordinating function.
- Use support entails a 1st and 2nd line help desk. The 1st line is a telephonic help desk and the second line involves application developers/functional managers.
- Request fulfillment is separated from incident management.
- ICT operations management and ICT operations control are performed by both the SaaS provider and the IaaS provider. Some tasks require interplay between them and some tasks can be performed by one of them.
- Monitoring can be performed at the IaaS provider and at the SaaS provider.

DIFFERENCES

- Service catalogue management is not applicable when only one application is offered. When multiple applications are offered, it should be managed.
- Requirements analysis could be a good process where customer and SaaS provider are involved. This process focuses on demand-driven supply.
- Change evaluation does not add extra value when evaluation is integrated in a mature process for multi-client change management. When only one application is managed, it may be valuable to evaluate separately, before initiating the change management process.
- Software management will still be necessary for cloud applications that are not multi-tenant and therefore need to be copied to multiple servers when using the application more than once. The customer, however, will not see any difference because this is dealt with by the SaaS or PaaS provider.
- Problem management may in the case of single-application management be integrated in the incident management process. When multiple applications are managed, it may be worthwhile to appoint a problem manager who analyzes recurring issues and issues occurring with multiple customers.

4.6. THEORY MODIFICATION

Based on the cross-case conclusions, the theory (SaaS management framework concept 1, change management concept 1 and configuration management concept 1) has been modified. This results in a second concept which can be found in appendix I. The modifications are shortly explained in this paragraph.

THE FRAMEWORK

The change with most impact on the framework is the decision to divide the service design, service transition and service operation phases into three layers. From the case studies it turned out that most processes were applicable, but not for every actor in the supply chain of services. A distinction is made between customer, service & platform management and infrastructure & hardware management. In both case studies, these were three separate parties. Side note is that the PKN chose to be their own “partner” for service and platform management. The difference between SaaS and PaaS vendors is subtle, since both are in essence software providers. A platform is also software and therefore the same processes are applicable.

In service strategy no changes have been made since the strategy comes before the decision to use SaaS deployed or model-driven developed applications.

In the service design phase no processes have been changed.

In the service transition phase, the processes *Change Evaluation*, *Configuration Management (service components)* and *Software Management* have received a question mark. These processes were inconclusive after both case studies, since one case did use these processes and the other did not. The process *Configuration Management (virtual and physical systems)* was moved from service operation to service transition since it seemed misplaced in the first concept. ITIL also places configuration management in the service transition phase.

In the service operation phase, the process *Problem Management* has been marked with a question mark since its necessity as a separate process was contradictory in both case studies. Its necessity was therefore inconclusive after conducting the case studies.

In continual service improvement, the process *Service Lifecycle Management* has been moved from service transition because of its overall presence in the lifecycle of a service and its role in all parties of the supply chain.

CHANGE MANAGEMENT

The actor “service level manager”, the process “event management” and documentation in the form of legislation and third party requirements are added to the input for the change management process.

The activities “Filter and approve RfC”, “Impact analysis and priorization” and “resource planning” have been added.

The decision point for urgent changes has been added and the accompanying activity with “urgent procedures”. The “transformation” activity has been removed since it is an automated part of the modeling activity. The “does it work?” decision point received an extra back-out line to be able to go back to the impact analysis activity.

The software management process had been added for it is subsequently connected to the deployment activity when a software/release management process is in place. After closing the change, the users may be notified about the changes.

CONFIGURATION MANAGEMENT

Since the configuration management process is not specifically defined in both cases, it may be concluded to be less important in a service environment where one or relatively few services are managed. It is quite conceivable that configuration management becomes more important when the range of services is larger and it is not defined for a few configuration items.

Since Mendix is developing a support tool for keeping the service catalogue and the expectation is that a future version of the Mendix platform will support the use of different components/modules, a large part of configuration management may be automated in the future.

Cross-case analysis is inconclusive on configuration management and therefore the process has not been modified. This creates the opportunity to address the configuration management process with all its activities in the framework validation phase of this study.

4.7. CHAPTER SUMMARY

In this chapter, the multiple case study approach was explained and conducted. Two cases have been studied, one at the PKN and one at Nobel. The PKN is an early user of the Mendix platform and currently deploy the largest Mendix application. Nobel is a service organization that develops services and takes care of hosting and management. Since the first only deploys and manages one service, it is referred to as single-client management. The second is a multi-client management organization. This affects some aspects of the management processes.

The data collection in the case studies focuses at eight general aspects amongst which the distinction and integration between process actors and the automation of processes. Also the practical use of the framework and configuration and change management processes were tested. The results of the interviews held in the case studies and documents were reviewed, after which the case conclusions have been draft. The conclusions of both case studies did not always match and were therefore found inconclusive on some aspects of the theory. All conclusive aspects were assimilated in the theory and the inconclusive aspects were appended with a question mark. The most significant change in the framework was the introduction of three different layers across the service design, service transition and service operation phases. These layers represent the customer, the service & platform managing organization and the infrastructure & hardware managing organization. For the processes, it does not make a difference whether these three layers are represented by one, two, three or more organizations in practice, since it indicates at which level processes are needed. Most management aspects are only needed in one or two of these layers. The theory modifications have been based on the cross-case analysis and can be found in appendix I.

5. FRAMEWORK VALIDATION

In the previous chapter the theory has been tested in practical environments and adapted to be effective in those settings. To validate the new theory as a whole, a review round will be conducted in which the second concept of the theory is discussed with PostNL and Capgemini Application Outsourcing experts. These two case studies will lead to validation of the framework, the configuration management process and the change management process. The validation discussion has been carried out in a structured way that is similar to the way the case studies were approached in the previous chapter. The difference is, however, that the questions which do not directly focus on the review of the theory are left out. Therefore there is no detailed list of questions; the aspects of the framework and the two processes have been reviewed one by one.

5.1. VALIDATION COMPANY C – POSTNL

This validation case study starts with introducing the case organization and the application(s) whose management has been studied. Next, the case study conclusions are presented. The interview reports used to conclude on this case study can be found in Appendix J.

COMPANY INTRODUCTION

The organization

TNT N.V. split in May 2011 into a mail company and an express company. The first was named PostNL and the latter is known as TNT Express. PostNL N.V. is a public limited liability company with its main office in 's-Gravenhage in the Netherlands. The company is listed on the NYSE Euronext in Amsterdam. It had a total operating revenue of 4,297 million euro and a net profit from continued operations of 213 million euro in 2011 (PostNL N.V., 2012).

PostNL focuses at the market of mail, parcel and e-commerce in the Netherlands, the UK, Germany, Italy and Belgium. The organization increasingly offers direct support for e-commerce by offering ready-made webshops or by managing the complete logistics process for online retailers. The postal company was originally established in 1799 by the Dutch government. With the first Postal Act in 1807 the government awarded the organization exclusive rights to collect, convey and deliver mail (PostNL N.V., 2011).

The Mendix platform is used by PostNL to be able to quickly build new applications for web and mobile for the benefit of their e-business activities (van Oeveren, 2012). PostNL is a typical customer organization in the classification of IaaS – SaaS – Customer.

The application

Capgemini has built an application called CVS ("Client Volg Systeem") for PostNL on the Mendix platform. This application exists of multiple modules that were developed in 9 releases. It enables job mobility and career support and it is used by the Mobility department of PostNL. The application replaced a traditional application to guarantee the continuity of operations and to be prepared for upcoming activities (Capgemini Talent NL, 2012). PostNL makes use of the Capgemini Agile/Mendix factory and the Salesforce SaaS.

REVIEW CONCLUSIONS

The SaaS management framework

- Contract management will shift more to functional use aspects and continuity aspects. The importance and dependency from the SaaS provider becomes more significant since the processes are external, stacked on one or multiple service providers. Making a contract and then securing the conditions will integrate more and more since the frequency and detail level increase. Multiple traditional internal functions are now integrated in one service contract.
- Service catalogue management is related to automated provisioning upon user request. The bill is connected to the provisioning environment.
- Subcontractor management includes the aspect of provider compliancy. The SaaS provider should make sure he and his subcontractors meet the customer agreements on compliancy. Compliance monitoring should be organized within the processes, not separately.

- The IT Service continuity management process is important for IaaS and SaaS provider and the customer. It is about how to act when a service breaks down.
- Planning and control is divided into separate processes at the different organizations. The organization will use the planning of their provider as input for this process, since SaaS is more supply-driven.
- Change evaluation is a provider process that checks with the customers whether the current functionality is sufficient. The focus becomes utilizing the SaaS capabilities. The process could also be called “Customer success management”.
- Software management is necessary in a cloud environment, because distribution is still in order. If the cloud is not multi-tenant, distribution is definitely applicable.
- Knowledge management should be used to explain the user how the service should function. It is actually the help file of the service catalogue and therefore it has a link to the customer. Customer self-service and guidance is important in a cloud environment since the provider cannot help everyone learn to work with the system.
- Access management: the infrastructure or service provider should not be able to simply login to the production environment. The SaaS provider delivers the ability to define users and roles, but the customer should be the one who creates them.
- Use support also has a link with the customer. The customer has a role in this process.
- Request fulfillment: the need for requests should in the first place be prevented by automation. Although, a request fulfillment ticket window is needed.
- Problem management should be practiced by the service provider. Recurring incidents still exist and should be solved.

Change management

Change management for the customer is different than it used to be, because it is supply-driven now. The customer just gets the new release with accompanying changes. The SaaS will be upwards-compatible and the customer will get more functionality in a new release. Change management for the customer entails whether it would be useful to start using the new functionality. This is more of a business change process instead of an IT process.

At the SaaS provider, a traditional change management process will be in place. This is demand-driven, but never for one organization, always for the entire market. The requirements will be determined generically.

Configuration management

A service provider will have configuration management to keep track of his service provisioning and the associated billing.

Configuration management will take place at both application- and hardware level. At application level, functionalities are configured and made into one working entity. The rationale is in the re-use of services.

5.2. VALIDATION COMPANY D – CAPGEMINI

This validation case study starts with introducing the case organization and the application(s) whose management has been studied. Next, the case study conclusions are presented. The interview reports as well as the documentation used to conclude on this case study can be found in Appendix K.

CASE INTRODUCTION

The organization

Capgemini division Custom Software Development (CSD) is a large partner of Mendix. The partnership contract was signed in September 2011 and both companies work close together to finish successful projects. The Mendix powered “Capgemini Agile Factory” followed up on existing SharePoint and .NET factories. (Capgemini Talent NL, 2011; Kepinski, 2012).

Ron Tolido, Capgemini CTO of Applications Continental Europe called applications developed on the Mendix platform ’scooters’ that increase the flexibility of the enterprise. Large IT systems developed with SAP or Oracle

are little flexible 'train connections' that are not quickly adaptable to the changing world outside (van Oeveren, 2012).

The application

Capgemini is a typical organization that provides management as a service. The organization therefore manages multiple applications, amongst which the former mentioned CVS for PostNL and applications for NS HiSpeed and Energie Beheer Nederland (EBN). Capgemini develops and manages SaaS applications based on platforms of Mendix as well as Salesforce and Be Informed. Capgemini deploys and manages its own solution "RM3" both as SaaS and on-premise.

REVIEW CONCLUSIONS

The SaaS management framework

- The strategic processes are above the application or deployment choice and therefore do not differ from traditional processes.
- Service catalogue management can be seen as the Mendix App store. All generic apps built by different companies are displayed here. The live services of a customer are all in one environment and can be seen by the developers.
- IT service continuity management is important at both the service & platform and the infrastructure & hardware. Its domain differs, but both domains need to communicate.
- The resources needed by the application are controlled by service managers, not the infrastructure managers. Resource management is therefore an interplay between the SaaS and IaaS providers. Resources are also the combination of human and skills.
- Planning and control are actually three separate processes. The customer can trigger adjustment of the base planning with incidents or change requests. It is both demand- and supply-driven since Capgemini performs monthly checks to find improvements to propose to the customer.
- A "proactive advice" process could be added to the SaaS level. This is about advising the customer about performance and convenience improvements when opportunities are seen to do so.
- Knowledge management describes the flow and coherence of the main processes, but not too much in detail since details are too susceptible to changes and outdated documentation is a risk. The details can be concluded from the model. In case of an exception the modeler should add a comment. Knowledge management is there for the developer, and documentation should actually be determined by the end-user.
- Change evaluation is mainly applied by the customer, not the SaaS provider. The customer tells the SaaS provider what to build. The process can be a kind of customer change board.
- Software management or distribution is important for non-multi-tenant services like the services that are built on the Mendix platform. Mendix applications can be installed on multiple locations.
- Use support is performed both at the customer and at the SaaS provider. End-users will contact the support desk at the customer. The functional managers or key-users will try to help the end-user and if they are unable to help, it will be passed on to the SaaS provider.
- Requests occur. Request fulfillment is therefore important, but it can be integrated in the use support process. Agreements with the customer can be made beforehand on the priority requests should get.
- IT chain continuity management becomes more important as a system integrator process. It involves knowledge on how the chain of a business process is built up; which applications are involved, in which order, what do they do and who is responsible.
- Access management: the SaaS provider creates user roles as part of the model and the functional managers of the customer create users and assign them to a role.
- Problem management plays only a small role in practice because of agile development. When incident management is not able to solve the incident, it is usually passed to change management. Problem management should be implemented at the SaaS, PaaS and IaaS providers.
- Monitoring and diagnostics can be performed by both the IaaS and the PaaS.
- Accounting and billing is present at all companies involved in the supply chain.
- Service lifecycle management is present at all participating organizations, but mainly at the SaaS provider.

Change management

The change management input comes from the customer. The filtering, impact analysis and prioritizing and informing the end-user are done by the customer. For example for PostNL, monthly a list of changes is reviewed and developed. Adjustments are weekly reviewed with the customer, so they can still be altered in the same change. Change management has overlap with the customer, but not with the IaaS provider. Planning is a separate activity and urgent procedures are definitely in place.

The development order is as follows: modeling a part, testing the part, after two or three parts the iteration is finished and deployed in a test environment, the customer will test it in the test environment after which it will be deployed in the production environment. A line back from “does it work” to impact analysis is unnecessary in practice. The impact analysis is not altered when something else is built than originally intended. Evaluation at the end is passed over in practice. The main difference in the change management process is that iterative development and customer involvement are better supported by model-driven development. There is a link to configuration management when the building starts; the new version is then created with the status “in development”.

Configuration management

Versions are managed at application level. The production server runs one version and the database with all former releases is delivered by Mendix. It is not possible to have different modules with different releases next to each other. The version number of modules that are used from the app store should be tracked for future support. The platform version is also tracked to be able to say whether a feature is available.

There is a configuration plan. Naming conventions are proposed by Mendix. There is no status monitoring or verification. Version management is automated by the Mendix platform.

Configuration management on the Mendix platform is minimal. A list of 5 to 10 constants can be set on instance-level.

5.3. THEORY MODIFICATION

Based on the expert reviews in companies C and D, the theory (SaaS management framework concept 2, change management concept 2 and configuration management concept 2) has been modified. This resulted in a third and final version. The framework is presented in this paragraph and the two processes can be found in appendix L. The modifications are explained shortly in this paragraph.

THE FRAMEWORK

The *Planning and Control* process has been broken up into separate processes. This emphasizes that every organization in the framework pursues its own planning and has its own control mechanisms. Communication between the different organizations at the level of this process is important so that the organizations in the chain can respond to developments at the other organizations. The communication goes both ways and is represented by the arrows.

Resource management has been found playing a role on the service & platform management level. Human resources (people and skills) need to be managed, but also service resource requirements need to be managed. The service provider may instruct the infrastructure & hardware provider to allocate more resources when found necessary by the service provider.

The *Change Evaluation* process is more of a business change process than an IT change process. When the SaaS provider adds more features to the service, the business should decide whether these features may be used to support business processes and possibly disrupt the business’ current way of working. Change evaluation may be a supply-driven process when using a centralized multi-tenant SaaS. Currently it is, however, not uncommon that an application referred to as SaaS is actually a distributed, hosted application. In that case traditional *Change Management* is in order and communicates with a change board or evaluation board within the customer organization. This requires communication back and forth, indicated by the arrows.

Configuration Management was found applicable, although it is only performed at the level of application and platform (no components are registered) and only the name and version are registered.

Software Management was also found applicable in hosted distributed (platform-based) services, since they are distributed. In a non-distributed SaaS environment, software management involves keeping track of former releases to be able to revert.

The *Chain Management* process adds value to the management framework when multiple applications are connected and form an application landscape. Business processes carry out actions across multiple applications and can be disrupted when one application is unavailable or encounters an error. The chain management process is about having the knowledge ready about business processes and where the applications are involved, so that the application or connection causing the problem can be solved quickly and well-thought. A system integrator or provider could include interface management in this process.

The process *Customer Success Management* is about the service provider helping the customer to use as much as possible of the features provided by the service. The more features are used by the customer, the more value for money the service will provide and the more the customer is bound to the service. This increases the probability of the customer continuing the use of the service.

The *Request Fulfillment* process has been integrated into the *Use Support* process. Use support represents one contact point for the customer where all questions can be asked. The priority of questions and issues should be discussed with the customer when defining the use support service. The use support process has been moved to show the customer's involvement. It is very plausible for organizations to have an internal use support process that is close to the end-user and the business processes. Only when the internal use support is unable to answer a question or solve an issue, the provider's use support is contacted.

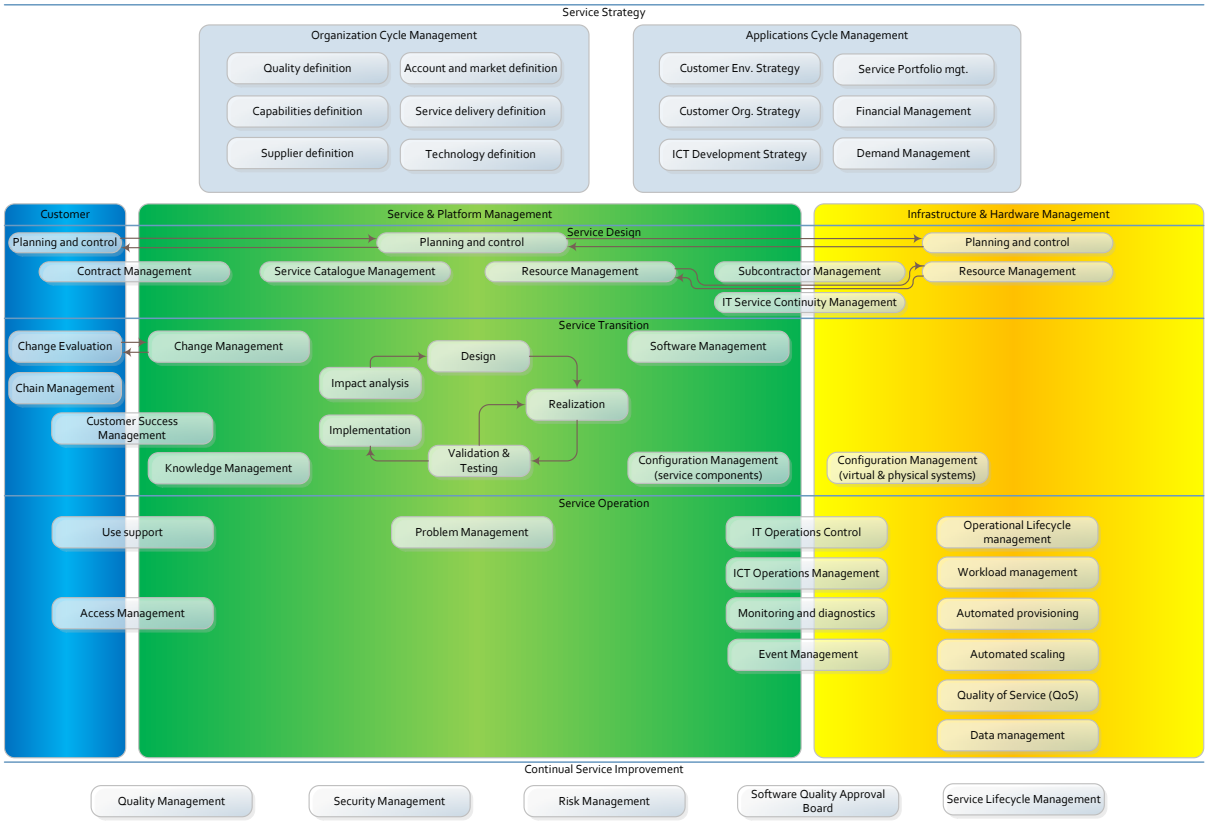


Figure 19: the final SaaS management framework

Problem Management should be implemented to handle recurring incidents and incidents that are experienced by many customers. In a model-driven developed service, the use of problem management will be reduced significantly since changes can be made quickly and the underlying building blocks of code have already been used many times before, which reduces the chance of them still containing structural errors.

Accounting and billing has been removed as a separate process. This is included in the strategic *Financial Management* process. The financial management process will be implemented at every organization that is involved and can be automated to a certain extent of choice.

CHANGE MANAGEMENT

In the change management process, a decision point is added to decide whether the change adds value to the market as a whole. This decision will be an issue when developing a standard service for many customers. The SaaS provider should always spend his development resources on improvements that serve the most customers. The resource planning activity has been moved before the urgent decision point since planning is no activity for urgent changes. The feedback line from “does it work?” to impact analysis is removed since the impact analysis is not affected when the modeler decides to develop the change differently. Finally, the Model-activity has been connected to configuration management since a new service/model version is created at that point.

CONFIGURATION MANAGEMENT

In the configuration management process, the change management process has been added as a process that provides input to the activities of registering and updating service configuration items. The activities of configuration item status monitoring and verification of the configuration management database/system have been removed. These activities become less important since the model development software keeps track of the current model and former model releases. These activities therefore gain a low priority and have not been implemented currently.

5.4. CHAPTER SUMMARY

The validation of the framework conducted through expert reviews was described in this chapter. Two distinct organizations have participated in the validation of which the approach was similar to the case studies in the previous chapter. PostNL and Capgemini were the organizations that participated in the validation. PostNL embraces the way applications are developed on the Mendix platform. The organization is typically on the demand side, while Capgemini is an IT supply-organization that develops services and takes care of the management. Capgemini has already successfully built an application on the Mendix platform for PostNL. PostNL is a typical “customer” in the SaaS framework and Capgemini is the SaaS provider. PostNL and Capgemini both manage multiple applications.

The data collection by expert review focused only at the framework processes and the change and configuration management processes in detail, case study questions on indirectly involved aspects were dropped. The practicality of the theory was tested by discussing it with people who are experienced in software development and management for both traditional and SaaS applications. The SaaS applications were represented by Mendix and Salesforce. The results of the interviews and optional documents were reviewed, after which the case conclusions have been drafted. This time, consistent conclusions could be drawn by combining all four case studies. Some aspects were mentioned in only one or two cases due to different maturity levels on certain aspects. Important changes in the framework are: division of the planning and control process, addition of the chain management and customer success management processes and the movement of the use support process to both the customer and the service & platform management organization.

The theory modifications resulted in the final SaaS management framework and process models of the two processes that were studied in more depth. The framework can be found in figure 19 and the process models in appendix L.

6. CONCLUSION AND DISCUSSION

The first paragraph of this chapter presents the conclusions to the main and sub research questions. The discussion paragraph elaborates on the research conduct and the validity of the research, after which the third paragraph discusses subjects and aspects that were not part of this study but could make relevant subjects in follow-up research. The fourth paragraph elaborates on the contribution to science and the relevance of this study in practice. The final paragraph, recommendations, provides an overview of the main advice to organizations that develop and manage a PaaS or SaaS application.

6.1. CONCLUSION

The main research question "How should the set of management and control processes in general be given shape for SaaS applications that were developed using model-driven technologies?" has been answered by studying five sub questions and ultimately in designing a framework and two processes as an artifact. The conclusions to the sub questions and the main question will be addressed one by one in this paragraph.

What are the distinguishing characteristics of cloud computing compared to traditional software?

Cloud computing has been defined as the on-demand provisioning of resources and services over a networked medium (Armbrust et al., 2010; N. A. Sultan, 2011). In principle cloud computing makes pooled resources available to any subscribing user, hardware resource usage is maximized by virtualization, resources are dynamically scalable up and down when needed, virtual machines can be created, deployed and deleted in an automated manner and resource usage is billed on per-use basis as opposed to sole upfront capital expenditure (Rosenberg & Mateos, 2011). Cloud computing entails three different service models and four deployment models (Mell & Grance, 2011). The cloud is service oriented, which means that software is abstract and accessible, loosely coupled and has a strong fault tolerance. Cloud services are easy to use; the cloud model reduces information technology overhead for the end-user. Other characteristics are: TCP/IP based, virtualization and high security (Gong et al., 2010). In other terms, Surgient (2009) endorses a dynamic computing infrastructure, IT service-centric approach, self-service based usage model, minimally or self-managed platform and consumption-based billing as the most important characteristics.

How do model-driven and agile development differ from traditional development methods?

MDA is the concept of models playing a direct role in software production (Kent, 2006). The definition of MDA states that the specification of functionality is separated from the specification of its implementation (Object Management Group, 2007). MDE is a wider term, complementing MDA with a combination of process and analysis. MDE involves the actual creation of an abstract model that is to be subsequently transformed systematically to create concrete software (France & Rumpe, 2007). MDE (or visual programming) can be seen as a fifth-generation programming language since it is based on solving problems by solely defining constraints to the program, rather than a programmer writing an algorithm (Schmidt, 2006). The two key differences between traditional and MDE are that the business analyst actually makes the changes to the application model and that underlying platform specific models and source code are generated automatically (ADA Software, 2012).

There are generally two types of software development: iterative and incremental methods that are commonly referred to as "agile" (Highsmith & Cockburn, 2001) and the "waterfall method" (Royce, 1970). Agile implies effectiveness and maneuverability by close collaboration with the end-user participating in the development process (Cockburn, 2001; Highsmith & Cockburn, 2001), as opposed to using the waterfall method, where the company would function as a filter to the end-users. Agile results in a more innovative and valuable system (Conboy & Morgan, 2011). Four values that describe how agile development differs from the more traditional waterfall method are: individuals and interactions are more important than processes and tools, working software is more important than comprehensive documentation, user collaboration is more important than contract negotiation and responding to change is more important than following a plan. An agile way of working saves costs since the end product fits the business better and changes are still possible in a relatively late development stage against lower cost than when using the waterfall method (Pressman, 2009).

What are effective methods for operational management of traditional IT applications?

The ITSM method provides directives to operationalize the concept of IT management. Supporting ITSM, ITIL is the worldwide de-facto standard (Behr et al., 2004; Larsen et al., 2006). Both the ITIL and ASL frameworks are generally mentioned when discussing IT governance models (Larsen et al., 2006) since they do include processes to cover all three business levels of strategic, tactical and operational execution (Souer et al., 2008). The ITIL and ASL frameworks provide best practices, measurement, improvement and management processes (Sallé, 2004; Shahsavarani & Ji, 2011). ITIL v3 consists of five books that represent the service lifecycle, and several complementary publications (Kempter et al., 2012). Knapp (2011) mentions a selection of 24 ITIL processes that can be seen as its main processes. While the focus of ITIL is at both technical infrastructure and service management, ASL is a collection of best practices with the focus on managing application development and maintenance. It is the public domain standard for application management and complementary to ITIL (Larsen et al., 2006; Smalley & Meijer, 2006). All ASL processes are structured in a recognizable framework (van der Pols & Backer, 2006). The Capgemini WoW framework for application management was originally created by combining the two most commonly used industry frameworks ITIL and ASL and extending it with CMMI and requirements for standards like ISAE3402, ISO27001 and ISO9001.

How should traditional frameworks for IT management and maintenance be adapted to provide effective directions in a model-driven SaaS environment?

The level of the ITIL framework is kept abstract on purpose, with the rationale that it can be used to shape any IT management environment. It should therefore be ready to work with new, emerging delivery models, associated to cloud computing for example. The strategic process layer will not be affected by the choice for a traditional application or cloud service since these decisions are made above the operational management. On the tactical and operational levels, processes will need to be altered or disappear, while new processes will start playing a role in the management model. Proposed differences in ASL can be found by comparison of figures 13 and 15 (van der Pols, 2008). Some management tasks can be automated, among which the provisioning of system images, performance and availability monitoring, metering and billing, monitoring and integration of applications (Smith, 2010). Smart scalability can also be automated (Rodero-Merino et al., 2010). DMTF efforts focus on SLA's, Quality of Service (QoS), workload portability, automated provisioning and accounting and billing (DMTF Inc., 2009). The Cloud Data Management Interface (CDMI) standard supports CRUD activities as well as discovery of cloud storage offering devices, manages data and its containers and allows metadata to be associated with data and containers (SNIA, 2011).Technical management of the hardware-components and physical network will be performed by the infrastructure-provider, excluding OS management. Cloud servers are always distant and therefore more management and maintenance has to be performed on the communication-infrastructure. Another aspect is integration with the company's existing systems and between services. Service integrators or cloud service brokers may have an important role therein (Tardijn, 2011). The main difference in ITIL incident management is in direct collaboration with the SaaS provider to investigate and solve any incidents (Guo & Wang, 2009). The combination of theory and the practical usability which has been tested and validated by case studies shows the ultimate answer to this sub question. The final framework is presented in figure 19.

How can the adapted management framework be developed and validated for practical use?

The design science research methodology was followed in this research to design and analyze the framework and processes as novel artifacts (Kuhn, 1970; Vaishnavi & Kuechler, 2004). There are several evaluation methods for the proposed design, the method used in this study has been *observational*; building the theory incrementally on what Hevner et al. (2004) refer to as 'the environment', justified by case/field studies. The case study approach itself has been based on the method described by Yin (2003), which is visualized in figure 17. Two case studies were conducted to test the framework developed from literature and the validation involved two different case studies with a similar approach. In the validation, the emphasis was on expert opinions, while the main emphasis in the testing phase was on current practices.

In what way are the change and configuration management processes affected?

The change management process of a multi-tenant SaaS is supply-driven from the customer's point of view. Changes are based on the demand of the market as a whole and not on a single-customer demand. Because of this significant difference, the customer could use a change management or evaluation process. The provider's

process is very similar to the traditional change management process since there is still software being developed and the process still deals with users who require changes or extra functionality. The customer's process is a whole new process. It will focus on the changes and extra functionality the customer receives in a new release of the service. The customer organization should explore the new functionality and the business managers, not the IT managers, should decide whether it is useful and should be adapted. The process at the service provider is an IT change process, involving software development, but the customer's process really is a business change process, scrutinizing and improving the business's way of working. Mendix is distributed, single-tenant and therefore the change management process for Mendix applications is traditional, demand-driven. When considering Mendix or model-driven development in general, the most important difference in the process is the agile way of developing the change. The development of the change exists of multiple iterations in which the customer is closely involved. Model-driven development supports a close collaboration between the developer and the end-user (who represents the customer) because of the rapidity by which a change can be implemented and evaluated.

Important in configuration management is to document the service version and the platform version in case of SaaS development on a distributed platform. Also registering the version of any generic components used in the service proved good practice. There is currently no need for recording more detailed characteristics. When using a platform like Mendix, version management can be integrated as an automated feature. The source (code or model) is saved in a database and changes or releases can be tracked. This makes it easy to revert to a former release. When the development software (either integrated or not integrated into the platform) keeps track, there is no need and pressure to manually maintain a configuration database. Configuration management for a multi-tenant SaaS customer involves setting variables and configuring the application to the organization's requirements. This does not involve registration of service characteristics in a database.

CONCLUSIONS TO THE MAIN RESEARCH QUESTION

An important difference between SaaS and traditional software is that the likelihood that multiple vendors and sub vendors are involved is higher. This is due to the relative easiness of starting to use a service. The governance model should take this into account and the framework is therefore split into three different types of actors. Currently there are solutions that claim to be SaaS but are actually a hybrid form between traditional software and SaaS. A distinction can be made between those two types. With a multi-tenant point-solution SaaS application, the traditional application management shifts to the SaaS vendor. A system integrator might then be a new party who connects the customer's SaaS and on-premise applications with each other, monitors the application landscape and keeps the overview. With a distributed SaaS solution, the application can be managed by the party who traditionally manages the organization's applications and systems. This means that the management framework can be implemented fluidly; the processes should be performed, but which party performs them depend on the application. The developed framework can therefore be implemented in hosted as well as SaaS environments.

The line between SaaS and PaaS vendor is currently blurred, since many PaaS vendors are actually developing a distributed application that supports the creation of applications on top of it. The platform is installed on-premise, which means that the used concept is not really SaaS and the vendor actually is an application vendor. Mendix is also presented as PaaS while it has characteristics of a hosted platform: it is distributed and can be deployed on any server environment. The platform supports multiple applications to run on top of it, but it does not provide the pre-developed building blocks to make these applications multi-tenant. The Nobel case study revealed that using an innovative term, which is actually a way of marketing, attracts more customers: the cloud-seminars are well-attended while hosting-seminars that involve the same subjects do not receive the same amount of attention.

6.2. DISCUSSION

In total, four organizations have been studied on their current way of managing model-driven developed SaaS and the theory has been reviewed by experts. All these organizations manage one or multiple services deployed on the Mendix platform and have experience in the management of traditional applications. Two of these organizations also develop software on the Mendix platform. Mendix development is in principle model-driven and the applications are delivered over a networked medium, presented as SaaS. The case organizations have been selected to represent many differences, e.g. in size, single- or multi-client management, customer or developer/integrator, types of applications and application size. The organizations are independent and

employees were interviewed independently. A case study protocol has been used to ensure an equal approach of every case and therefore increase the reliability and validity of the research. A case study database has been set up to be able to browse and search evidence in later stadia. The chain of evidence between the collected data and conclusions in the report has been maintained as much as possible to be able to track all steps taken (Benbasat, Goldstein, & Mead, 1987; Yin, 2003). To increase the case study validity, the quantitative technique of data triangulation has been used, involving different sources of information (Guion, Diehl, & Mcdonald, 2011; Olsen, 2004; Tellis, 1997).

Because the applicability to other cases cannot be proved, the external validity of a case study research is typically low. However, to maximize the external validity of the study, the case studies have been based on existing literature. This means that in principle, proven concepts are addressed (like ITIL and ASL) that are already being used in daily processes in a wide range of different industries and companies.

Every organization has its own way of working. It is therefore difficult to compare processes on detail-level. The first organization will tell to add an activity or feedback-line that the next organization tells to remove. However, the processes are in general the same amongst organizations, but also amongst deployment and even development techniques. The SaaS management framework has been generalized to support model-driven as well as template-driven or code-driven development and hosted as well as SaaS deployment. Also the aspect of single- or multi-management processes has been taken into account. The framework can be generalized to all these aspects. Frameworks like ITIL and ASL were developed to be generic and should be molded into a specific management situation for a specific application. Every organization is different, requires different activities, decisions and documentation. Each organization emphasizes different activities from their own specific approach. It does therefore make sense that the framework resulting from this study does not diverge much from the traditional framework. However, when developing a framework or a process it can never be assumed to work as well for not yet existing uses, so existing frameworks should always be tested in new situations.

6.3. LIMITATIONS AND FURTHER RESEARCH

This study can be seen as a foundation in the research area of SaaS management. The framework has been developed that instructs on the definition of management on a high abstraction level. The framework processes were not all developed in-depth. In-detail development based on good practices is however useful in practice and therefore very interesting for organizations. Mainly the processes that differ significantly from the processes in traditional management (e.g. financial management, chain management, customer success management, access management) are suitable for further research. The use of CMMI maturity levels may be more applicable on the detail execution of processes than on the framework as a whole.

The emphasis in this research has been on Mendix, which means that other development and deployment platforms have been underexposed. There are other emerging platforms and tools that support model-driven development and there are even more platforms that support SaaS deployment. This study has shown that many current SaaS applications are actually distributed, hosted applications and therefore the developed framework took both into account. Further research may focus on the different requirements between management of distributed and centralized SaaS applications.

Within the scope of this research it has not been able to discuss the framework with SaaS platform developers like Mendix, Be Informed and Salesforce nor with vendors that focus specifically on cloud infrastructure and hardware management. These aspects were tested by the interviews and documentation of SaaS organizations that perform software development and infrastructure and hardware management in-house. These aspects have therefore been underexposed in the research. Also management of data, people, projects and of the processes itself are topics for follow-up research.

The processes in the service strategy and continual service improvement fell outside the scope of this research because of the expectation that these phases in the service lifecycle would not be affected by SaaS deployment or model-driven development. There may however be (subtle) differences that need to be taken into account when making use of these new techniques.

6.4. RELEVANCE AND CONTRIBUTION

The effects of SaaS deployment on the management as a whole has not been studied and published before, however, Capgemini did study the effects of more applications being offered as a service. This research expands the current scientific research base on the subject of SaaS management and provides a foundation for more detailed research to follow up, since the specific process flow of most processes has not been studied. The effect of model-driven development on the change management process has not been scientifically published, but can be found on-line. The contribution to science on this subject is in the evaluation of the change management process for model-driven development.

This research contributes to practice since many organizations have started developing or adopting SaaS or are thinking about adopting SaaS but are unaware of the consequences for the organization of their management. This study provides a mainstay for the processes needed in cloud service management. It is not only useful for service providers, but also customers may use it to verify whether their provider is in control.

6.5. RECOMMENDATIONS

IT management frameworks like ITIL and ASL are developed to be generic and to be shaped to the specific business and application situation. They are based on best practices for traditional applications and can therefore not be assumed to fit cloud solutions as well. This research shows that management at the abstraction level of the framework is still very similar; the activities still need to be performed. However, some activities are different in detail or they may be performed by a vendor. The SaaS management framework provides a guideline that is developed to be generic and therefore abstract. It is always recommended to be critical and implement a generic framework with common sense, as it is for the existing frameworks like ITIL and ASL. A framework should fit the organizational needs and every organization is different. It should therefore be shaped and fine-tuned to the specific situation.

The introduction of cloud services entailed a new spectrum of roles an intermediate organization like Capgemini could fulfill. These are for example new emerging support roles like cloud services strategist, broker, integrator, auditor or certifier (Dreyfuss, 2010). However, the case studies show that traditional roles like developer or consultant are still very current. The SaaS management framework shows which management processes are applicable at which level in the supply chain. For each application, the line between provider, sub provider and customer may differ. A system integrator takes care of two things: integrating and connecting applications and development of custom software. The integrator should focus on intimacy at the customer side and at industrialization and innovation at the vendor side.

An issue with SaaS deployment is that it is difficult to use a selected group of actual end-users as pilot testers before implementing a change to everyone. The options are limited to the test and production environments. Testing can only be done in the test environment with test data. It is not able to synchronize the edits users make here with the production environment.

MODEL-DRIVEN AND AGILE DEVELOPMENT

Agile model-driven development quickly produces results and the connected processes should anticipate that. For example the procurement process of the customer takes much time and is therefore not ready for agile development. When the customer sees how quickly it can be built, he expects more to be built immediately. Writing decent documentation and composing a suitable management contract is then omitted because that takes more time. These aspects and the consequences of leaving them out should be discussed at the start of a project. For agile software development, the distinction between the management processes is more blurred; consultants are all-round and can be deployed on every aspect of the development. The Mendix framework does not support the creation of application documentation. The model is actually its own documentation. However, in specific situations the original model developer should have had the discipline to describe (e.g. in a comment) why he has made a specific choice. An advantage of non-agile development methods is that the system design documents can be used in practice as the system's documentation after its development.

It might be an option to create standardized templates of common processes modeled in a tool like Mendix. These templates could be based on best practices and with some modifications for the specific customer

deployed for that customer. This saves development time and bugs in the model (since the standardized model is used before, bugs may already have been eliminated).

GOOD PRACTICES

When an organization exists of distributed divisions that have their own system administrator, the divisions cannot be separated in the system to be individually backed up and restored. A solution can be creating a test environment which is being refreshed with actual data monthly. This way, the local administrator and users can test the system response to certain actions or test new functionality in a harmless way.

The easiness by which additional functionality can be developed with the Mendix framework is of course an advantage, but it also has a downside. It should be avoided to make (small) changes to the system that are not analyzed or documented well, because then there is no control of the process. All changes should be evaluated and then structured implemented. When issues and requests are submitted well described in a standard template, this relieves the application managers and shortens the agile development process because less iterations of development and testing are necessary. The requester should be included in the iteration's testing activity to make sure that it was developed the way it was meant.

The Mendix platform does not support read-only or "observer" authorizations. Such authorizations would be convenient to show development progress to people who are not allowed to make changes but do have to account for it or benefit by being able to see the progress.

The traditional incident management process could be used for "agile management". However, the delivered functionality of the application should be documented clearly enough before it can be transmitted from development to management. Uniformity of development is an interesting aspect to understand the developed models better.

REFERENCES

4CaaSt. (2011). *Blueprinting the Cloud: Scientific and Technical Report*. (Y. Taher, Ed.).

ABN AMRO. (2012). Groeistuipen van Cloud Computing. Retrieved April 26, 2012, from www.abnamro.nl/nl/images/Generiek/PDFs/020_Zakelijk/02_Sectoren/Media_en_Technologie/media-rapport-cloudcomputing.pdf

ADA Software. (2012). Model driven architecture [MDA]. Retrieved August 2, 2012, from www.adasoftusa.com/menu4-mda.html

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods*. Espoo, Finland: Otamedia Oy.

April, A., Huffman Hayes, J., Abran, A., & Dumke, R. (2005). Software Maintenance Maturity Model (SMmm): the software maintenance process model. *Journal of Software Maintenance and Evolution: Research and Practice*, *17*(3), 197–223. doi:10.1002/smr.311

Armbrust, M., Fox, A., Griffith, R., Stoica, I., Zaharia, M., Joseph, A. D., Katz, R., et al. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58. doi:10.1145/1721654.1721672

Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., & Neema, S. (2006). Developing Applications Using Model-Driven Design Environments. *Computer*, *39*(2), 33–40. doi:10.1109/MC.2006.54

Be Informed. (2012). Be Informed. Retrieved February 17, 2012, from www.beinformed.com

Behr, K., Kim, G., & Spafford, G. (2004). *The visible ops handbook: implementing ITIL in 4 practical and auditable steps*. Information Technology Process Institute.

Beimborn, D., Miletzki, T., & Wenzel, S. (2011). Platform as a Service (PaaS). *Wirtschaftsinformatik*, *53*(6), 371–375. doi:10.1007/s11576-011-0294-y

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly*, *11*(3), 369–386.

Bittman, T. J. (2011). The Road Map From Virtualization to Cloud Computing. *Gartner RAS Core Research Note G00210845*. Retrieved from www.gartner.com/id=1572031

Boland, R. J. (1979). Control, causality and information system requirements. *Accounting, Organizations and Society*, *4*(4), 259–272. doi:10.1016/0361-3682(79)90017-5

Boynton, A. C., Zmud, R. W., & Jacobs, G. C. (1994). The influence of IT management practice on IT use in large organizations. *Mis Quarterly*, *18*(3), 299–318.

Breiter, G., & Behrendt, M. (2009). Life cycle and characteristics of services in the world of cloud computing. *IBM Journal of Research and Development*, *53*(4), 3:1–3:8. doi:10.1147/JRD.2009.5429057

CMMI Product Team. (2002). *Capability Maturity Model Integration (CMMI SM), Version 1.1*. Pittsburgh, PA: Engineering Institute (SEI), Carnegie Mellon University.

COSO. (1992). Committee of Sponsoring Organizations of the Treadway Commission (COSO). *Internal Control - Integrated Framework*. Retrieved May 30, 2012, from www.coso.org

Capgemini. (2011). Annual Report 2011. Retrieved June 11, 2012, from www.capgemini.com/annual-report/2011/en/download/Capgemini-RA2011-INST_EN.pdf

Capgemini Talent NL. (2011). Capgemini Forges Ahead with Mendix. Retrieved November 15, 2012, from http://talent.capgemini.com/nl/news/10747

Capgemini Talent NL. (2012). Agile Factory PostNL. Retrieved November 15, 2012, from http://talent.capgemini.com/nl/news/12008

Cartlidge, A., Hanna, A., Rudd, C., Macfarlane, I., Windebank, J., & Rance, S. (2007). *An Introductory Overview of ITIL V3*. (A. Cartlidge & M. Lillycrop, Eds.)*The UK Chapter of the* (1st ed.). UK: itSMF Ltd.

Chang, V., Walters, R. J., & Wills, G. B. (2012). Business Integration as a Service. *International Journal of Cloud Applications and Computing*, *2*(1), 16–40. doi:10.4018/ijcac.2012010102

Cockburn, A. (2001). *Agile software development*. Boston, USA: Addison-Wesley Professional.

Conboy, K., & Morgan, L. (2011). Beyond the customer: Opening the agile systems development process. *Information and Software Technology*, *53*(5), 535–542. doi:10.1016/j.infsof.2010.10.007

Cordys B.V. (2012). Platform as a Service. Retrieved June 11, 2012, from www.cordys.com/platform-as-a-service

DMTF Inc. (2009). Virtualization MANagement (VMAN): A Building Block for Cloud Interoperability. Retrieved July 2, 2012, from http://dmtf.org/sites/default/files/VMAN-Cloud Interoperability Overview_2010.pdf

DMTF Open Cloud Standards Incubator. (2009). Interoperable Clouds. Retrieved July 2, 2012, from http://dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf

DMTF Open Cloud Standards Incubator. (2010). Architecture for Managing Clouds. Retrieved July 2, 2012, from http://dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf

Day, B., & Lutteroth, C. (2011). Climbing the ladder: capability maturity model integration level 3. *Enterprise Information Systems*, *5*(1), 125–144. doi:10.1080/17517575.2010.495789

Delen, G. P. A. J., & van Looijen, M. (1992). *Beheer van de informatievoorziening*. Rijswijk: Cap Gemini Publishing.

Diversity Analysis. (2010). Software Delivery Models: Comparing Approaches and Debunking the Myths.

Dolstra, E., Bravenboer, M., & Visser, E. (2005). Service configuration management. *Proceedings of the 12th international workshop on Software configuration management - SCM '05* (pp. 83–98). New York, New York, USA: ACM Press. doi:10.1145/1109128.1109135

Dreyfuss, C. (2010). Strategy , Architecture and Governance in Cloud Services Management, 2010-2011, (November 2010).

Ebbers, M., O'Brien, W., & Ogden, B. (2006). *Introduction to the New Mainframe: z/OS Basics*. (IBM, Ed.) (1st ed.). IBM.

Eisenhardt, K. M. (1989). Building theories from case study research. *The Academy of Management Review*, *14*(4), 532–550.

France, R., & Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering (FOSE '07)* (pp. 37–54). IEEE. doi:10.1109/FOSE.2007.14

Gong, C., Liu, J., Zhang, Q., Chen, H., & Gong, Z. (2010). The Characteristics of Cloud Computing. *2010 39th International Conference on Parallel Processing Workshops* (pp. 275–279). IEEE. doi:10.1109/ICPPW.2010.45

Guion, L. A., Diehl, D. C., & Mcdonald, D. (2011). Triangulation: establishing the validity of qualitative studies. *University of Florida*, 1–3.

Guo, W., & Wang, Y. (2009). An Incident Management Model for SaaS Application in the IT Organization. *2009 International Conference on Research Challenges in Computer Science* (pp. 137–140). IEEE. doi:10.1109/ICRCCS.2009.42

Hassanzadeh, A., Namdarian, L., & Elahi, S. (2011). Developing a framework for evaluating service oriented architecture governance (SOAG). *Knowledge-Based Systems*, *24*(5), 716–730. doi:10.1016/j.knosys.2011.02.012

Heliview Research. (2012). Gebruik cloud computing verdubbeld ondanks weerstand. Retrieved April 26, 2012, from www.heliview.nl/website.nsf/0/C04E39ABBB7086D3C12579AD00383450

Hemsoth, N. (2010). Mainframe, Client-Server and the Clouds Above. *HPC in the Cloud*. Retrieved May 25, 2012, from www.hpcinthecloud.com/hpccloud/2010-12-01/mainframe_client-server_and_the_clouds_above.html

Henderson, J. C., & Venkatraman, N. (1993). Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, *32*(1), 4–16.

Henkel, M., & Stirna, J. (2010). Pondering on the Key Functionality of Model Driven Development Tools: The Case of Mendix. In P. Forbrig & H. Günther (Eds.), *Perspectives in Business Informatics Research* (Vol. 64, pp. 146–160). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-16101-8

Henry, B. (2007). Programming language generations. Retrieved June 14, 2012, from http://searchcio-midmarket.techtarget.com/definition/programming-language-generations

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. *Computer*, *34*(9), 120–127. doi:10.1109/2.947100

Hofland, M., & Molendijk, A. (2011). *Way of Working Application Outsourcing Netherlands*.

IBM. (2008). Align IT with business goals using the IBM Process Reference Model for IT.

Iwasa, K. (2011). IT Management SaaS. *Fujitsu Scientific and Technical Journal*, *47*(3), 335–341.

Jha, S. K. (2012). ITIL and Cloud Computing. *ITILnews.com*. Retrieved August 13, 2012, from www.itilnews.com/ITIL_and_Cloud_Computing_by_Sumit_Kumar_Jha.html

Kempter, S., Kempter, A., & Lea-Cox, T. (2012). IT Process Wiki: The ITIL Wiki. Retrieved June 21, 2012, from http://wiki.en.it-processmaps.com/index.php/Main_Page

Kent, S. (2006). Model Driven Engineering. In M. Butler, L. Petre, & K. Sere (Eds.), *Integrated Formal Methods* (Vol. 2335). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/3-540-47884-1

Kepinski, W. (2012). Partners zijn cruciaal voor Mendix. *Dutch IT-channel*. Retrieved November 7, 2012, from www.dutchitchannel.nl/dic/3/26/3666/_partners_zijn_cruciaal_voor_mendix_.html

Kieskamp, H. (2011). Project LRP. *Presentation held on the Mendix World 2011 Conference*. Retrieved October 17, 2012, from http://www.slideshare.net/HHKieskamp/mendix-world-pkn-28apr2011-0v2c

Knapp, J. (2011). *ITIL V3 Foundation Exam Video Mentor*. (B. Brown, S. Schroeder, C. Betz, B. Hopper, & V. Evans, Eds.). Indianapolis, Indiana, USA: Pearson Education, Inc.

Krikhaar, R., & Crnkovic, I. (2007). Software Configuration Management. *Science of Computer Programming*, *65*(3), 215–221. doi:10.1016/j.scico.2006.10.003

Kuhn, T. S. (1970). *The Structure of Scientific Revolutions*. (O. Neurath, R. Carnap, & C. Morris, Eds.) (2nd ed., Vol. II). The University of Chicago.

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments: a brief history. *Computer*, *36*(6), 47–56. doi:10.1109/MC.2003.1204375

Larsen, M., Pedersen, M., & Viborg Andersen, K. (2006). IT Governance: Reviewing 17 IT Governance Tools and Analysing the Case of Novozymes A/S. *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)* (Vol. 00, p. 195c–195c). IEEE. doi:10.1109/HICSS.2006.234

Lawton, G. (2008). Developing Software Online With Platform-as-a-Service Technology. *Computer*, *41*(6), 13–15. doi:10.1109/MC.2008.185

Leavitt, N. (2009). Is Cloud Computing Really Ready for Prime Time? *Computer*, *42*(1), 15–20. doi:10.1109/MC.2009.20

Lenk, A., Klems, M., Nimis, J., Tai, S., & Sandholm, T. (2009). What's inside the Cloud? An architectural map of the Cloud landscape. *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing* (pp. 23–31). IEEE. doi:10.1109/CLOUD.2009.5071529

Lheureux, B. J., Thompson, J., Skybakmoen, T., & Wilson, D. R. (2010). Predicts 2011: Application Integration: A Nimble Veteran of IT. Retrieved from www.gartner.com/id=1476026

Lientz, B. P., & Swanson, E. B. (1980). *Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations*. Reading MA: Addison-Wesley Publishing Company.

Lin, G., Fu, D., Zhu, J., & Dasmalchi, G. (2009). Cloud Computing: IT as a Service. *IT Professional*, *11*(2), 10–13. doi:10.1109/MITP.2009.22

Lindquist, D., Madduri, H., Paul, C. J., & Rajaraman, B. (2007). IBM Service Management architecture. *IBM Systems Journal*, *46*(3), 423–440. doi:10.1147/sj.463.0423

Little, M. (2008). The Generic SOA Failure Letter. *InfoQ*. Retrieved June 20, 2012, from www.infoq.com/news/2008/11/soa-failure

Loh, L., & Venkatraman, N. (1992). Determinants of information technology outsourcing: a cross-sectional analysis. *Journal of management information systems*, *9*(1), 7–24.

Louden, K. C., & Lambert, K. A. (2012). *Programming Languages: principles and practice*. (M. Lee & B. Shailer, Eds.) (3rd ed.). Boston, USA: Course Technology.

Louridas, P. (2010). Up in the Air: Moving Your Applications to the Cloud. *IEEE Software*, *27*(4), 6–11. doi:10.1109/MS.2010.109

Madison, J. (2010). Agile Architecture Interactions. *IEEE Software*, *27*(2), 41–48. doi:10.1109/MS.2010.35

Managementboek. (2012). Remko van der Pols. Retrieved August 10, 2012, from www.managementboek.nl/auteur/6179/remko-van-der-pols

Marinos, A., & Briscoe, G. (2009). *Cloud Computing*. (M. G. Jaatun, G. Zhao, & C. Rong, Eds.)*Cloud Computing* (Vol. 5931, p. 11). Networking and Internet Architecture; Distributed, Parallel, and Cluster Computing; Software Engineering, Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-10665-1

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing — The business perspective. *Decision Support Systems*, *51*(1), 176–189. doi:10.1016/j.dss.2010.12.006

Marten, H., & Koenig, T. (2011). Availability measurement of grid services from the perspective of a scientific computing centre. *Journal of Physics: Conference Series*, *331*(6). doi:10.1088/1742-6596/331/6/062038

McKinney, A. (1991). A Report on Computer-Aided Software Engineering (CASE). *Journal of Computing Sciences in Colleges*, *6*(5), 85–91.

McNurlin, B., Sprague, R., & Bui, T. (2009). *Information Systems Management in Practice* (8th ed.). Prentice Hall.

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing - Recommendations of the National Institute of Standards and Technology. *NIST Special Publication 800-145*.

Mendix. (2012). Product | Mendix | Next-generation Business Applications Made Easy. Retrieved February 17, 2012, from www.mendix.com/product

Mendix Technology B.V. (2011). *Mendix Factsheet*. Rotterdam.

Midha, V., & Bhattacherjee, A. (2012). Governance practices and software maintenance: A study of open source projects. *Decision Support Systems*. doi:10.1016/j.dss.2012.03.002

Miller, H. G., & Veiga, J. (2009). Cloud Computing: Will Commodity Services Benefit Users Long Term? *IT Professional*, *11*(6), 57–59. doi:10.1109/MITP.2009.117

Miller, J., & Mukerji, J. (2001). Model Driven Architecture (MDA). *ORMSC/2001-07-01*.

Mohan, K., Xu, P., Cao, L., & Ramesh, B. (2008). Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems*, *45*(4), 922–936. doi:10.1016/j.dss.2008.03.003

Niessink, F., Clerc, V., Tijdink, T., & van Vliet, H. (2005). *The IT Service Capability Maturity Model v1.0, Release Candidate 1*.

Niessink, F., & Vliet, H. V. (1998). Towards mature IT services. *Software Process: Improvement and Practice*, *4*(2), 55–71. doi:10.1002/(SICI)1099-1670(199806)4:2<55::AID-SPIP97>3.0.CO;2-T

Niessink, F., & van Vliet, H. (2000). Software maintenance from a service perspective. *Journal of Software Maintenance: Research and Practice*, *12*(2), 103–120. doi:10.1002/(SICI)1096-908X(200003/04)12:2<103::AID-SMR205>3.0.CO;2-S

OCCI. (2011). Open Cloud Computing Interface. Retrieved July 3, 2012, from http://occi-wg.org/

Object Management Group. (2007). *Business Process Model and Notation (BPMN) 2.0*.

Olsen, W. (2004). Triangulation in social research: qualitative and quantitative methods can really be mixed. (M. Holborn, Ed.)*Developments in sociology*, 1–30.

Papazoglou, M. P., & van den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, *16*(3), 389–415. doi:10.1007/s00778-007-0044-3

Papazoglou, M. P., & van den Heuvel, W.-J. (2011). Blueprinting the Cloud. *IEEE Internet Computing*, *15*(6), 74–79. doi:10.1109/MIC.2011.147

Paulk, M. C., Curtis, B., Chrissis, M., & Weber, C. V. (1993). Capability maturity model, version 1.1. *IEEE Software*, *10*(4), 18–27. doi:10.1109/52.219617

Pereira, R. F. de S., & Mira da Silva, M. (2010). A Maturity Model for Implementing ITIL v3. *2010 6th World Congress on Services* (pp. 399–406). IEEE. doi:10.1109/SERVICES.2010.80

Pereira, R., & Mira da Silva, M. (2011). A Maturity Model for Implementing ITIL V3 in Practice. *2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops* (pp. 259–268). IEEE. doi:10.1109/EDOCW.2011.30

Persberichten.com. (2011). Delta Lloyd en Nobel realiseren Expertise Service Portal. Retrieved November 7, 2012, from www.persberichten.com/persbericht.aspx?id=67618

Peterson, R. R. (2003). Integration strategies and tactics for information technology governance. In W. van Grembergen (Ed.), *Strategies for information technology governance* (pp. 37–80). Idea Group Publishing.

PostNL N.V. (2011). Our Company. Retrieved November 15, 2012, from www.postnl.com/about/company/index.aspx

PostNL N.V. (2012). Annual Report 2011. Retrieved November 15, 2012, from http://annualreport2011.postnl.com/introduction/charts-at-a-glance/show

Pressman, R. S. (2009). *Software Engineering: A Practitioner's Approach* (7th ed.). McGraw-Hill.

Rodero-Merino, L., Vaquero, L. M., Gil, V., Galán, F., Fontán, J., Montero, R. S., & Llorente, I. M. (2010). From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, *26*(8), 1226–1240. doi:10.1016/j.future.2010.02.013

Rollbase Inc. (2012). Rollbase: cloud platform for independent software vendors. Retrieved July 6, 2012, from www.rollbase.com/platform.shtml

Roos, D. (2010). Nobel closes strategic partnership with Mendix for Software-as-a-Service. *Mendix*. Retrieved November 7, 2012, from www.mendix.com/press/nobel-closes-strategic-partnership-with-mendix-for-software-as-a-service/

Rosenberg, J., & Mateos, A. (2011). *The Cloud at Your Service*. Greenwich: Manning Publications Co.

Royce, W. W. (1970). Managing the development of large software systems. *proceedings of IEEE WESCON*, (August), 1–9.

SNIA. (2011). Information Technology - Cloud Data Management Interface (CDMI ™). Retrieved July 3, 2012, from http://snia.org/sites/default/files/CDMI_SNIA_Architecture_v1.0.1.pdf

Salesforce.com. (2012). CRM-software on-demand | CRM (klantrelatiebeheer). Retrieved February 27, 2012, from www.salesforce.com/nl

Sallé, M. (2004). IT Service Management and IT Governance: review, comparative analysis and their impact on utility computing. *Hewlett-Packard Company*. Retrieved from www.itu.dk/courses/SISM/E2010/Salle_2004.pdf

Sanders, R. (2009). Ook nieuw systeem gaat PKN miljoenen kosten. *Computable*. Retrieved October 17, 2012, from http://www.computable.nl/artikel/nieuws/development/3087206/1277180/ook-nieuw-systeem-gaat-pkn-miljoenen-kosten.html

Schmidt, D. C. (2006). Model-Driven Engineering. *Computer*, *39*(2), 25–31. doi:10.1109/MC.2006.58

Schwalbe, K. (2011). *Information technology project management* (6th ed.). Boston, USA: Course Technology.

Shahsavarani, N., & Ji, S. (2011). Research in Information Technology Service Management (ITSM): Theoretical Foundation and Research Topic Perspectives. *International Conference on Information Resources Management (CONF-IRM)*.

Shaw, R., Chung, W., Cheng, C., & Fu, T. (2012). Key performance indicators for information technology (IT) operational management of financial industry in Taiwan. *African Journal of Business Management*, *6*(3), 1041–1053. doi:10.5897/AJBM11.2075

Simon, H. A. (1996). *The Sciences of the Artificial* (3rd ed.). Cambridge, MA: MIT Press.

Simonsson, M., & Johnson, P. (2006). Defining IT governance - a consolidation of literature. *the 18th Conference on Advanced Information*.

Smalley, M., & Meijer, M. (2006). *Frameworks for IT Management*. (J. van Bon & T. Verheijen, Eds.) (1st ed.). Zaltbommel, NL: itSMF-NL / Van Haren Publishing.

Smith, D. M. (2010). Hype Cycle for Cloud Computing, 2010, (July 2010).

Smith, R. (2009). InformationWeek Analytics: State Of SOA. *InformationWeek*. Retrieved July 2, 2012, from www.informationweek.com/news/214501922

Souer, J., Honders, P., Versendaal, J., & Brinkkemper, S. (2008). A framework for Web Content Management System operations and maintenance. *Journal of Digital Information Management*, *6*(4), 324–331.

Stahl, T., Völter, M., & Czarnecki, K. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

Strauss, A., & Corbin, J. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* (2nd ed.). London, UK: Sage Publications, Inc.

Sultan, N. (2010). Cloud computing for education: A new dawn? *International Journal of Information Management*, *30*(2), 109–116. doi:10.1016/j.ijinfomgt.2009.09.004

Sultan, N. A. (2011). Reaching for the “cloud”: How SMEs can manage. *International Journal of Information Management*, *31*(3), 272–278. doi:10.1016/j.ijinfomgt.2010.08.001

Sun, W., Zhang, K., Chen, S.-K., Zhang, X., & Liang, H. (2007). Software as a Service: An Integration Perspective. In B. J. Krämer, K.-J. Lin, & P. Narasimhan (Eds.), *Service-Oriented Computing - ICSOC 2007* (Vol. 4749, pp. 558–569). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-74974-5

Surgient, D. M. (2009). The five defining characteristics of cloud computing. *ZDNet*. Retrieved July 4, 2012, from www.zdnet.com/news/the-five-defining-characteristics-of-cloud-computing/287001

Tang, K., Zhang, J. M., & Jiang, Z. B. (2010). Framework for SaaS Management Platform. *2010 IEEE 7th International Conference on E-Business Engineering* (pp. 345–350). IEEE. doi:10.1109/ICEBE.2010.79

Tardijn, R. (2011). Hoe de cloud IT-beheer verandert. *IT-infra*, 14–17.

Tellis, W. (1997). Application of a case study methodology. *The qualitative report*, *3*(3).

The lifecycle company. (2012). Remko van der Pols. Retrieved August 10, 2012, from www.thelifecyclecompany.nl/index.php?/Onze-medewerkers/remko-van-der-pols.html

Vaishnavi, V., & Kuechler, W. (2004). Design Science Research in Information Systems. *January 20, 2004, last updated September 30, 2011*. Retrieved August 6, 2012, from http://desrist.org/desrist

Vaquero, L. M., Rodero-Merino, L., Caceres, J., & Lindner, M. (2009). A break in the clouds. *ACM SIGCOMM Computer Communication Review*, *39*(1), 50–55. doi:10.1145/1496091.1496100

Waters, B. (2005). Software as a service: A look at the customer benefits. *Journal of Digital Asset Management*, *1*(1), 32–39. doi:10.1057/palgrave.dam.3640007

Webb, P., Pollard, C., & Ridley, G. (2006). Attempting to Define IT Governance: Wisdom or Folly? *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)* (Vol. 00, p. 194a). IEEE. doi:10.1109/HICSS.2006.68

Weiser, M., & Brown, J. S. (1996). The coming age of calm Technology. In P. J. Denning & R. M. Metcalfe (Eds.), *Beyond calculation* (pp. 75–85). New York, NY, USA: Copernicus.

Welke, R., Hirschheim, R., & Schwarz, A. (2011). Service-Oriented Architecture Maturity. *Computer*, *44*(2), 61–67. doi:10.1109/MC.2011.56

Yin, R. K. (2003). *Case study research: Design and Methods* (3rd ed.). Sage Publications, Inc.

Young, C. M. (2004). An Introduction to IT Service Management. *Research Note COM-10-8287, Gartner*. Retrieved from www.gartner.com/id=295695

Young, C. M. (2011). Four key IT Service Management Frameworks. Gartner. Retrieved from www.gartner.com/id=1599635

de Haes, S., & van Grembergen, W. (2004). IT Governance and its Mechanisms. *Information Systems Control Journal*, *1*.

van Bon, J., de Jong, A., Kolthof, A., Pieper, M., Rozemeijer, E., Tjassing, R., van der Veen, A., et al. (2007). *It Service Management: An Introduction* (1st ed.). Zaltbommel: Van Haren Publishing.

van Bon, J., & van der Veen, A. (2007). *Foundations of IT Service Management - op basis van ITIL* (5th ed.). Zaltbommel, NL: Van Haren Publishing.

van Looijen, M. (2004). *Beheer van informatiesystemen*. Deventer: Kluwer.

van Oeveren, M. (2012). Scooters nemen het op tegen starre trein: Wordt Mendix de tweede TomTom? *DutchCowboys*. Retrieved November 15, 2012, from www.dutchcowboys.nl/software/26058

van Wamelen, J. (2010). Governance of IT in the network society. *Proceedings of the 4th International Conference on Theory and Practice of Electronic Governance - ICEGOV '10* (pp. 216–222). New York, New York, USA: ACM Press. doi:10.1145/1930321.1930366

van der Pols, R. (2001). *ASL - een framework voor applicatiebeheer* (1st ed.). Den Haag, NL: Academic Service, SDU Uitgevers bv.

van der Pols, R. (2008). *The Application Services Library, adapted to the IT-services world of the future*. Zaltbommel: Van Haren Publishing.

van der Pols, R., & Backer, Y. (2006). *ASL: A Management Guide*. (H. Boland, Ed.) (2nd ed.). Zaltbommel: Van Haren Publishing.

APPENDIX A: THE CAPGEMINI FRAMEWORK FOR IT MANAGEMENT PROCESSES

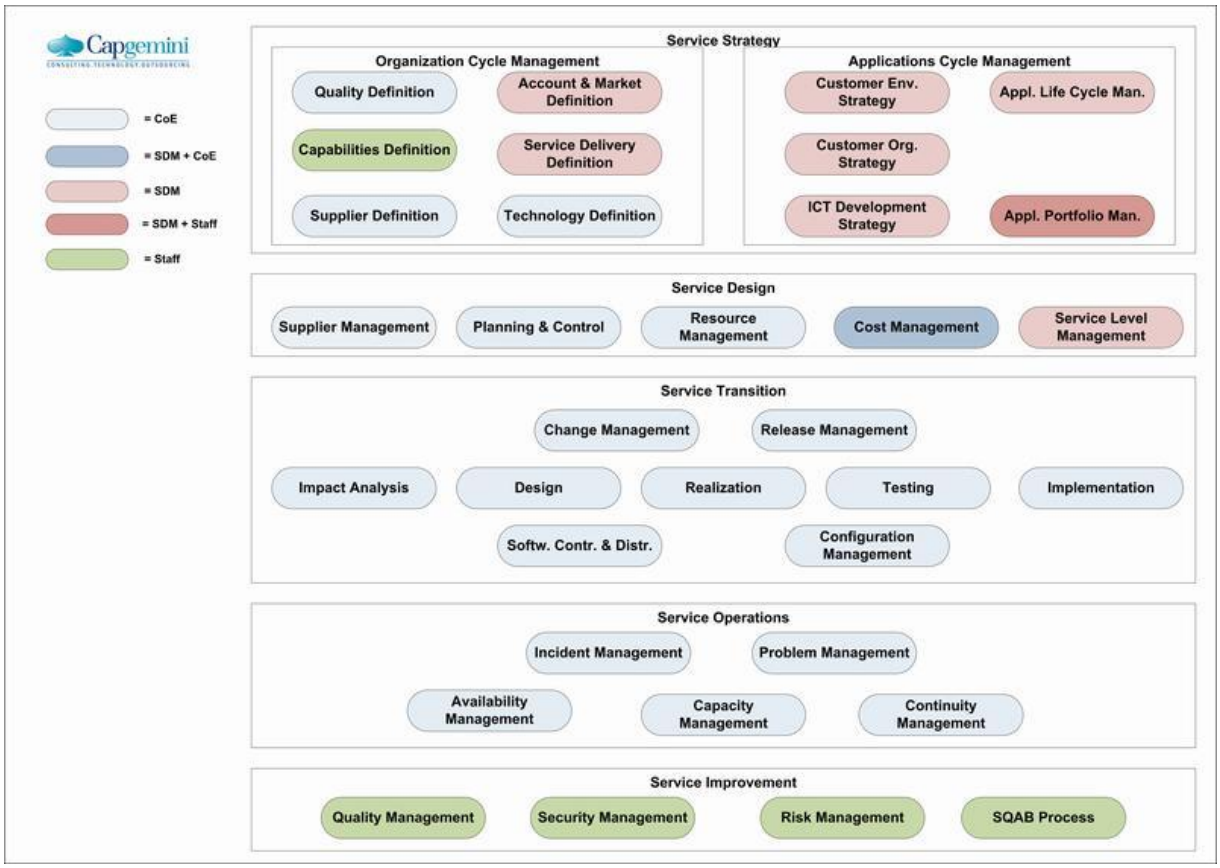


Figure 20: Capgemini NL Application Outsourcing framework "Way of working"

The model Capgemini Application Outsourcing (AO) refers to as "Way of Working" is both a description of AO services as well an in-depth framework for the overall maintenance process related to AO services. Development of this standard in-house methodology started in 2010 and it combines industry standards like ASL and ITIL with Dutch best practices while maintaining compliancy with Deliver (Outsourcing Roadmap).

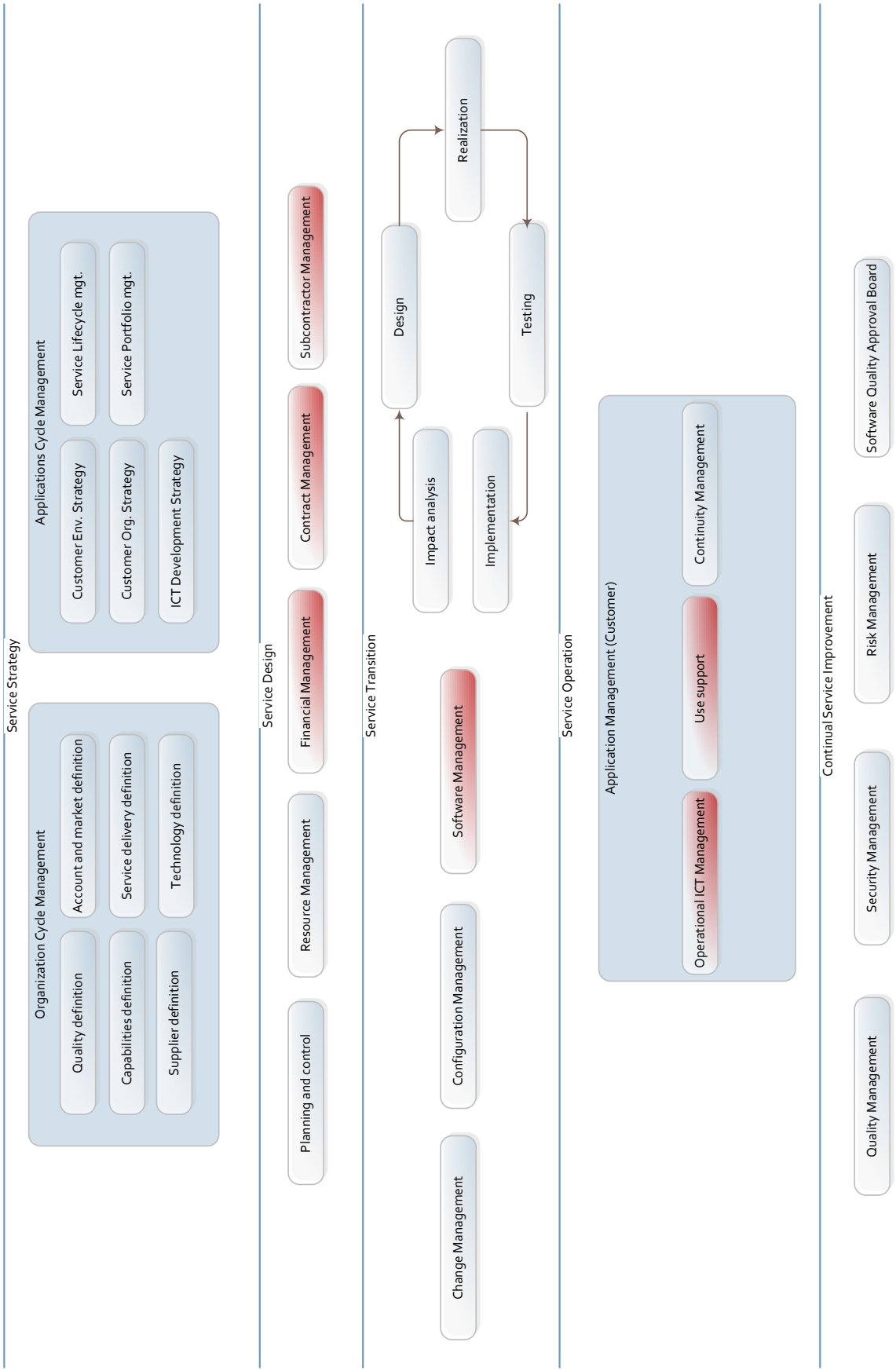
The development of this new standard was primarily focused on the integration of several well established methodologies into a Capgemini uniform set of procedures for both Outsourcing NL as well as our rightshore teams in India. The framework not only offers guidelines for the initial set up of services for new contracts and customers but it also facilitates the ongoing and continuous improvement process within our installed base.

As a company standard the "Way of Working" will serve as a generic toolset for all employees within the AO service domain providing a solid foundation for delivering "best of class" services to our customers. The model is continuously being improved based on user comments.

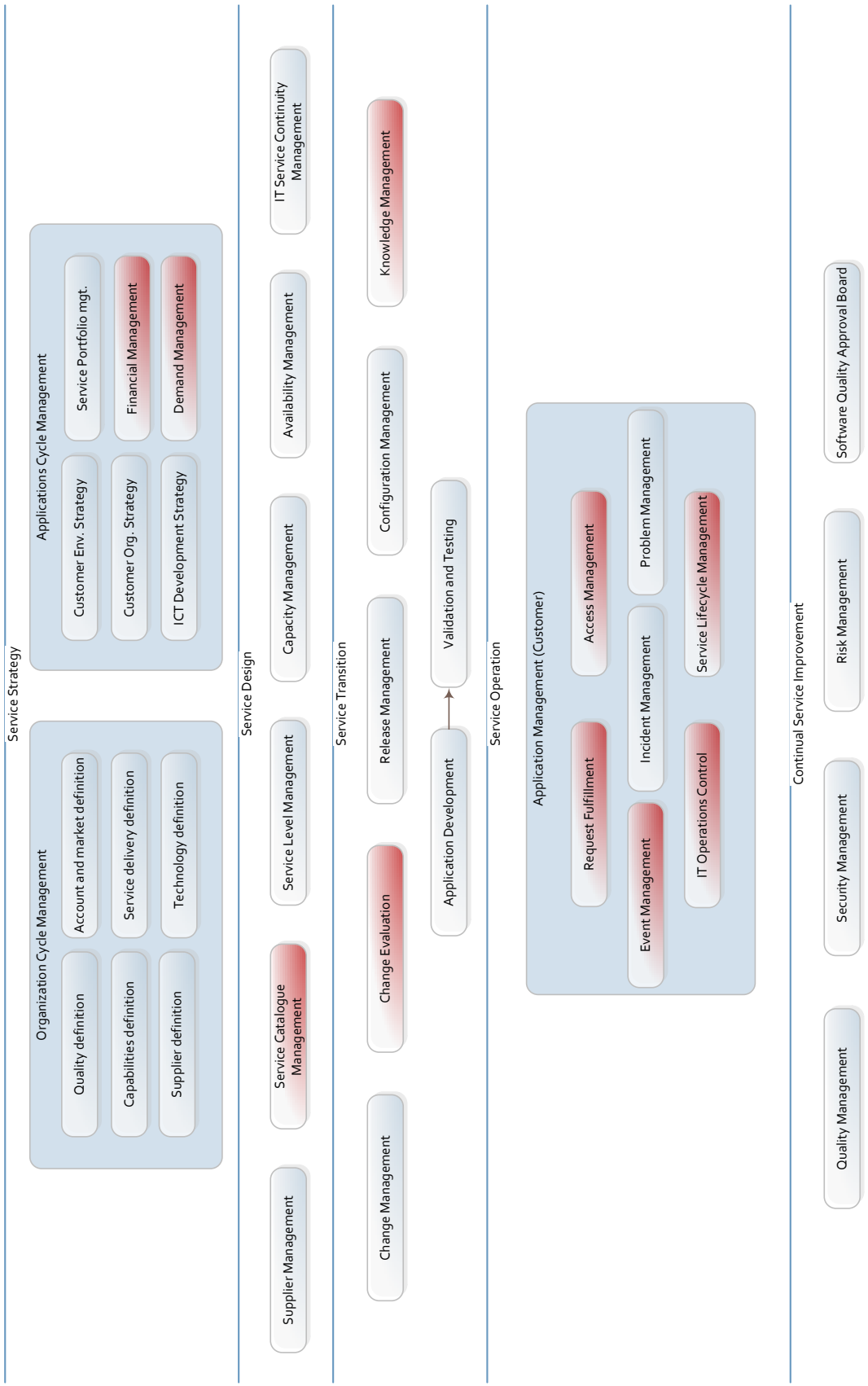
The Capgemini internal "Way of Working" site contains information on the organizational aspects within the framework, the basic principles and products (process descriptions, templates, etc.) underlying the methodology as well as detailed information about the process improvement and support processes. More technical information related to these subjects can be found in the individual process description documents.[1]

[1] Source: Capgemini (internal) "Knowledge Management 2.0" system

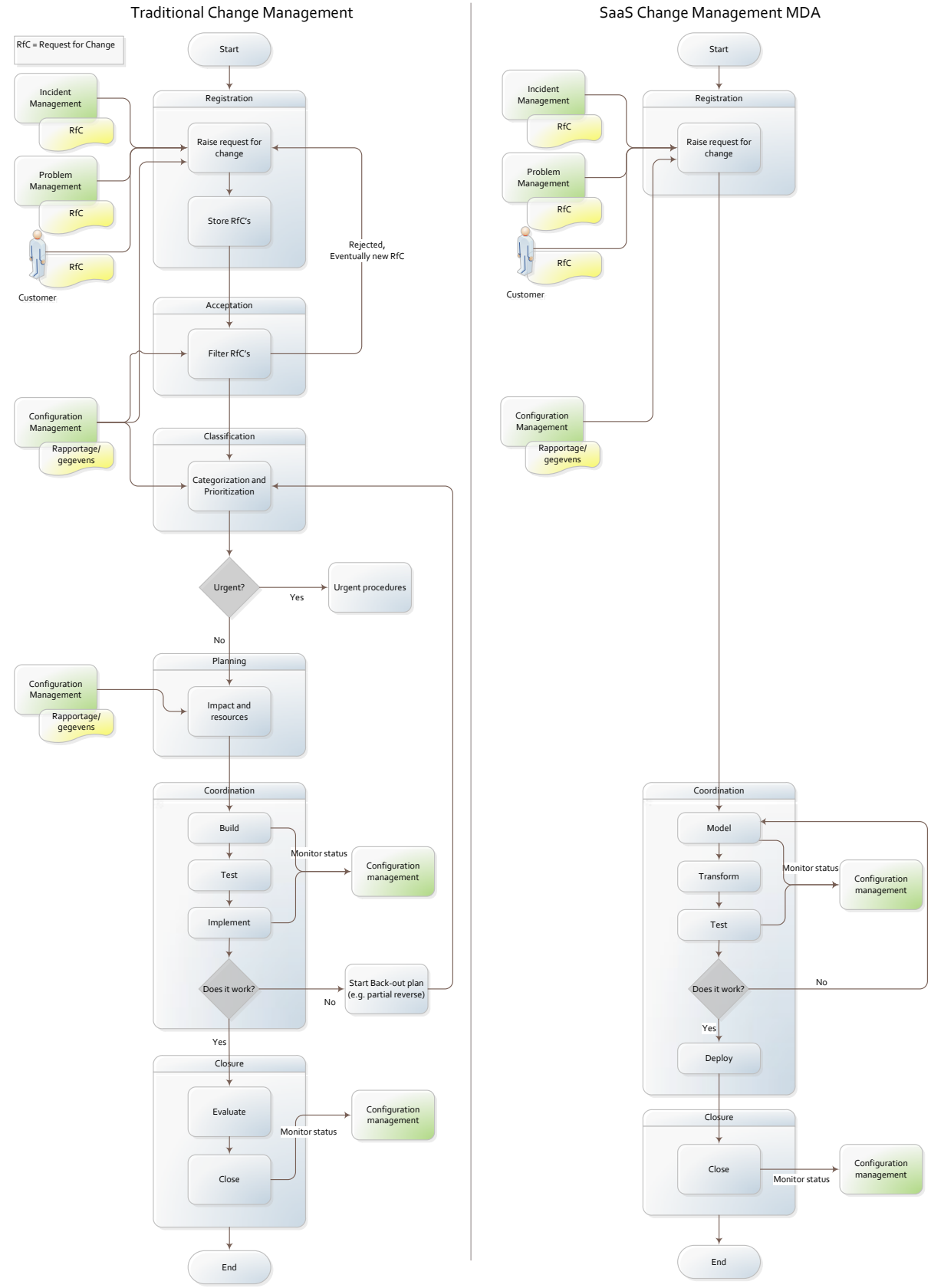
APPENDIX B: THE FRAMEWORK WITH ASL SAAS ADJUSTMENTS



APPENDIX C: THE FRAMEWORK WITH ITIL V3 ADJUSTMENTS

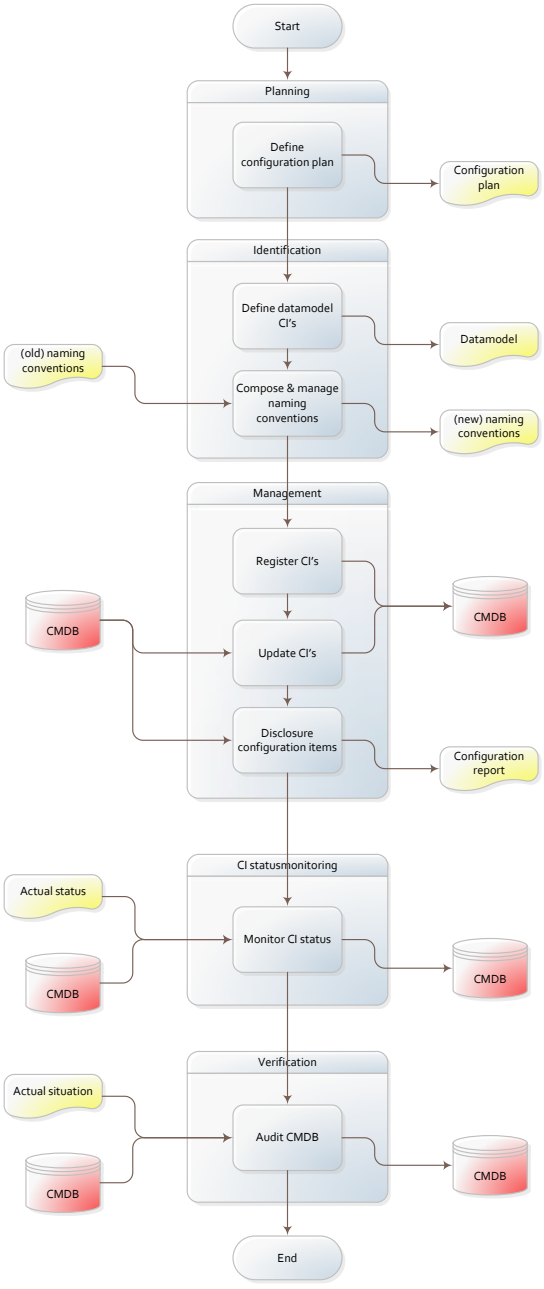


APPENDIX D: THE "CHANGE MANAGEMENT" PROCESS – CONCEPT 1

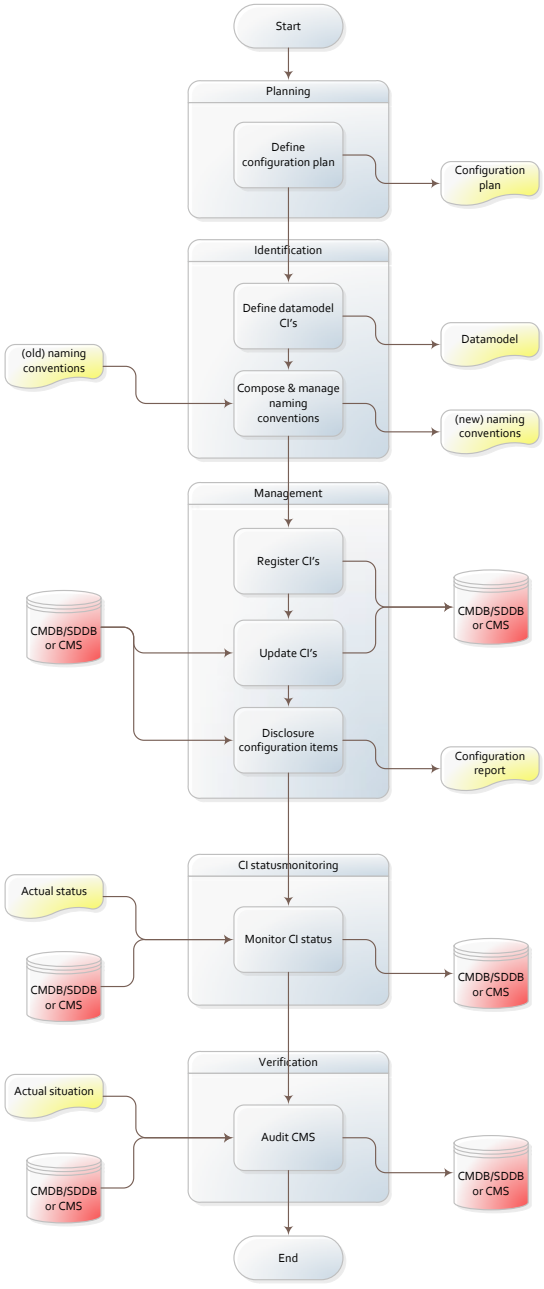


APPENDIX E: THE "CONFIGURATION MANAGEMENT" PROCESS – CONCEPT 1

Traditional Configuration Management



SaaS Configuration Management



CI = Configuration Item
CMDB = Configuration Management Database
SDDB = Service Delivery Database
CMS = Configuration Management System

APPENDIX F: CASE STUDY PROTOCOL

OVERVIEW OF THE STUDY

The case study will involve reviewing the conceptual framework at three organizations that have a Mendix application in use and actively perform a defined set of management tasks to maintain the application. The goal of these case studies is to optimize the alignment of the theory with practice. On the abstract level of the framework, processes can be ruled out or additional processes can be added when their goals are not met by other processes in the framework. On process-detail level, the configuration and change management processes need to be optimized for SaaS and model-driven development. The processes need to find a balance between control on the one hand and flexibility on the other hand.

FIELD PROCEDURES

CASE REQUIREMENTS

The cases need to comply with some requirements to fit the research and be selected. A case should have the following characteristics to be applicable:

- A Mendix application should be running within the organization
- The organization should actively manage this application
- There should be a recognizable set of management processes

The following characteristic is not mandatory but would definitely provide an advantage in understanding of the processes and communicating the differences of SaaS management:

- Experience with traditional application management

In consultation with Capgemini coworkers, the Mendix Partner Account Manager and online research, cases that comply with the requirements were selected. The following organizations will participate in the case studies/validation case studies:

- Protestantse Kerk in Nederland (PKN)
- Nobel
- PostNL
- Capgemini (Application Outsourcing division)

CONTACT

After the case study selection, the case site needs to be contacted. Since it is important to follow up quickly, the site will be contacted by phone and e-mail. The contacts named by the account manager will be contacted by phone to ask for the contact details of people who would be relevant to interview. An appointment for an interview with the relevant people will be made by phone. When the appointment has been made, the following e-mail will be sent with an attached invitation to inform the interviewee about the subject and remind them of the appointment. Also it will prepare the subject to save time in the actual interview.

Beste {Naam geïnterviewde},

Via {Mendix accountmanager} heb ik vernomen dat u betrokken bent bij het applicatiebeheer van {naam applicatie} en telefonisch heeft u aangegeven tijd te willen vrijmaken voor een interview. Alvast bedankt hiervoor!
Hierbij bevestig ik onze telefonisch overeengekomen afspraak op {datum + tijd} voor het interview.

Graag stel ik mezelf hierbij kort voor: ik ben Anne Slaa, ik studeer Information Management aan de Universiteit van Tilburg en ik ben momenteel bezig met mijn afstudeeronderzoek bij Capgemini. Het doel van mijn afstudeeronderzoek is het ontwikkelen van een praktisch raamwerk van IT beheerprocessen voor SaaS en modelgedreven ontwikkelde applicaties.
Om tot een raamwerk te komen dat niet alleen goed werkt in theorie maar ook in de praktijk voer ik een case study uit om het raamwerk te testen voor een drietal bedrijven. {naam bedrijf} is één van de

bedrijven waarvoor ik in kaart wil brengen hoe de set van managementprocessen eruit ziet en in detail hoe de processen configuration management en change management worden vormgegeven.
Door interviews met u en enkele collega's en het bestuderen van documentatie en procesbeschrijvingen wil ik een zo goed mogelijk beeld krijgen hiervan.

Op de volgende onderwerpen wil ik graag tijdens het gesprek ingaan:

- Het verschil tussen processen die {naam bedrijf} uitvoert en die een leverancier uitvoert en de integratie hiertussen
- Geautomatiseerde beheerprocessen
- De set van strategische processen
- De set van ontwikkelprocessen
- De set van overgangsprocessen
- De set van operationele processen
- De uitvoering van het configuratiebeheerproces
- De uitvoering van het veranderbeheerproces

De onderwerpen geven in hoofdlijnen aan waar ik het over wil hebben. Per onderwerp heb ik gedetailleerde vragen opgesteld die aan bod zullen komen. Mocht u vooraf nog vragen hebben dan hoor ik deze graag.

Nogmaals bedankt voor de tijd en tot {datum + tijd}.

Met vriendelijke groet,
Anne Slaa

RESOURCES

The required resources for an interview are:

- A separate room
- Notebook and pen
- Voice recorder
- Prints of the theoretical framework and conf. + change management processes

VERIFICATION

A report should be made after each interview. The report contains the summary and interpretation of what the interviewee said. The report will be e-mailed to the interviewee so that he/she can approve its content. This e-mail will be sent no later than the day after the interview to still have the interview subjects clearly in mind.

DOCUMENT THE COMMENTS

The comments on the theoretical framework and configuration and change management processes will be documented per category after each interview. This keeps the results organized and facilitates the interpretation at the end of each case study.

PLANNING SCHEDULE

After the collection of all data within a case study and approval of interview reports, the case study report should be written in five working days.

DATA MANAGEMENT

The case studies will be identified by company name in the final report, since no confidential data needs to be protected. All interviews will be numbered sequentially to refer to interviews without making statements directly traceable to a person. The author will keep a case study database containing all voice recordings, interview notes, all versions of the interview report and the documents retrieved from the field study so that all research steps can be traced back if necessary.

The final case report will be based on all the items that are described above. Every case will have an own report which will facilitate cross-case conclusions to be drawn in order to adjust the theory.

THE CASE STUDY INTERVIEWS

INTERVIEW INTRODUCTION

Allereerst wil ik u bedanken dat u de tijd heeft willen nemen voor dit gesprek. Graag zou ik mijzelf en mijn onderzoek kort introduceren voordat we met het interview beginnen. Mijn naam is Anne Slaa, ik volg de studie Information Management aan de Universiteit van Tilburg en ben bezig met het schrijven van mijn afstudeerscriptie. Mijn afstudeeronderzoek is voortgekomen uit de vraag van Capgemini of huidige methoden voor applicatiebeheer nog wel toereikend zijn wanneer applicaties op een modelgedreven manier worden ontwikkeld en vervolgens worden aangeboden via een platform in de cloud. De applicaties die ontwikkeld zijn op het Mendixplatform voldoen aan beide voorwaarden en zijn daarom erg geschikt voor de praktische benadering van de onderzoeksvraag. Graag zou ik nu met u spreken over de Mendixapplicatie die uw bedrijf in eigen beheer onderhoudt. Ik heb een aantal vragen uitgeschreven om richting te geven aan het gesprek en daarnaast kan ook de theorie ter ondersteuning dienen van de vragen. Indien u na het interview aangeeft dat u niet wilt dat uw naam of bedrijfsnaam in het eindrapport en eventuele verdere publicaties genoemd wordt zal dit geanonimiseerd worden.

INTERVIEW QUESTIONS

Vergelijking met eventuele voorgaande applicatie

- Heeft de applicatie een traditionele applicatie vervangen?
- Is er een beschrijving beschikbaar van de set van beheerprocessen voor traditionele app.?
- Is er een procesbeschrijving beschikbaar voor configuratiebeheer voor traditionele app.?
- Is er een procesbeschrijving beschikbaar voor veranderbeheer voor traditionele app.?
- Is er andere documentatie beschikbaar (functioneel ontwerp, templates, memo's, functieomschrijvingen, etc.)?

Applicatie

- Wanneer is de applicatie in gebruik genomen?
- Hoeveel tijd heeft de ontwikkeling gekost?
- Wat is het doel van de applicatie?
- Wat is de omvang van de applicatie in aantal functiepunten, gebruikers, departementen?
- Wie zijn de gebruikers van de applicatie?

Management

- Welke set van managementprocessen heeft X ingericht?
- Zijn deze managementprocessen beschreven?
- Is de interactie tussen managementprocessen beschreven?
- Op welke punten verschilt het beheer van de SaaS applicatie van een traditionele applicatie?
- Is management gebaseerd op één of meerdere standaarden?
- Wie is verantwoordelijk voor de beheerprocessen; welke actoren?
- Welke aspecten worden bewust niet beheerd?
- Welke aspecten worden door een externe partij beheerd (outsourcing)?
- Hoe is dit onderscheid afgedekt in de SLA('s)?
- Welke vrijheid heb je als afnemer/klant?
- Wat zijn de specifieke policies/randvoorwaarden voor het management zoals het nu is ingericht bij X?
- Zijn de beheerprocessen modelgedreven?
- Welke beheerprocessen/taken zijn geautomatiseerd en in hoeverre?
- Worden er periodiek rapporten gemaakt (reporting)? Zo ja, welke? Wat gebeurt er met deze rapporten?

SaaS management framework

- Is er een beschrijving beschikbaar van de generieke set van beheerprocessen bij X?
- Welke processen uit het framework concept zijn niet apart vastgelegd bij X?
- Welke processen heeft X vastgelegd die ontbreken in het framework concept?
- Welke actoren zijn er binnen het framework?
- Wie beslist over het framework?

Configuration Management

- Is er een procesbeschrijving beschikbaar voor configuratiebeheer?
- Komt de procesbeschrijving overeen met de uitvoering in de praktijk? Op welke punten wel/niet?
- Is er andere documentatie beschikbaar (templates, memo's, functieomschrijvingen, etc.)?
- Op welke punten is configuratiebeheer anders dan bij een traditionele applicatie?
- Welke delen van de IT-infrastructuur worden geregistreerd als CI?
- Welke informatie is daarvoor nodig?
- Welke servicecomponenten worden geregistreerd als CI?
- Welke informatie is daarvoor nodig?
- Welke actoren en verantwoordelijkheden zijn er binnen het proces?
- Wat zijn de specifieke policies/randvoorwaarden voor het proces?
- Wie beslist over het proces?

Change Management

- Is er een procesbeschrijving beschikbaar voor veranderbeheer?
- Komt de procesbeschrijving overeen met de uitvoering in de praktijk? Op welke punten wel/niet?
- Is er andere documentatie beschikbaar (RfC, templates, memo's, functieomschrijvingen, etc.)?
- Op welke punten is veranderbeheer anders dan bij een traditionele applicatie?
- Waar wordt change management op toegepast?
- Welke actoren en verantwoordelijkheden zijn er binnen het proces?
- Wat zijn de specifieke policies/randvoorwaarden voor het proces?
- Wie beslist over het proces?

CASE STUDY REPORT

The case study report should follow this structure:

- Company description
  - Company industry and branch
  - Company core activity
  - Company product(s)
  - Company size in employees/revenue/countries
  - Company strategy
  - Company competition/differentiation in its market
- Application description
- Findings SaaS framework
- Findings Configuration Management
- Findings Change Management
- Additional findings
- Positions of the interviewees
- The single interview reports (with aggregated answers)

The company description aspects will be researched upfront as a preparation to the interviews. The selection of appropriate interviewees will be done during the initial contact with the organization. The application description will be discussed on the basis of the interviews, as well as all the findings. The additional findings section contains the case study findings that are not directly related to the framework or the implications of SaaS or model-driven development, but are however interesting aspects to keep in mind when making the decision to develop/deploy a Mendix application. A recapitulation that is approved by the author of every single interview will be added to finalize the case report.

APPENDIX G: CASE STUDY A

To perform this case study, interviews have taken place and documents that fall within the context of the research subject have been collected. The introduction text of the case study refers to external sources which have additionally been used to prepare the case study and verify some of the subjects that were mentioned in the interviews.

Interviews		
Code	**Position**	**Date**
A-I1	Head of ICT and former project leader LRP	19-10-2012
A-I2	Application manager LRP	19-10-2012

Table 3: Interviews held in case study A

Documentation		
Code	**Title**	**Date**
A-D1	Automatisering Gids “Dan maar van scratch”	19-08-2011
A-D2	Bouw-test-en-implementatie-LRP-GPL process scheme	28-09-2012
A-D3	Handleiding Ledenadministratie - LedenRegistratie Protestantse Kerk versie 2.1	08-2012
A-D4	Handleiding Lokaal beheerder - LedenRegistratie Protestantse Kerk versie 2.1	07-2012
A-D5	PKN Folder dienstverlening	06-04-2011
A-D6	Training Bijdragenadministrateur powerpoint presentation	08-12-2010
A-D7	Training Ledenadministrateur powerpoint presentation	02-2010
A-D8	Training Lokaal Beheer powerpoint presentation	02-2011
A-D9	Processtappen-bugs-wensen-LRP-GPL process scheme	10-2012
A-D10	Specificatie-voor-LRP-wensen-en-verbetervoorstellen template	11-10-2012
A-D11	Mendix Support & Service Level Agreement for LRP version 3.0.4	15-11-2011

Table 4: Documentation used in case study A

The presentations and instruction manuals that are used as documentation can be found on the “LRP Netwerk” website http://LRP-netwerk.PKN.nl.
The more specific link to the manuals: http://lrp-netwerk.pkn.nl/30_Documentatie/30_Handleidingen

The interview reports are not literal transcriptions of the interviews, but mentioned aspects are partly summarized and shuffled to present a more logical story. As much information as possible is included in the interview reports to not accidentally overlook important aspects that may have gotten lost when summarizing. The digital recordings of the interviews as well as the mentioned documentation are available for inspection with the author.

INTERVIEW REPORT A-I1

Interviewee position: Head of ICT / Project leader LRP during implementation
Interview date: 19-10-2012
Interview duration: 94 minutes

Vergelijking met voorgaande applicatie(s)
Als je helemaal teruggaat in de geschiedenis was er een mainframeapplicatie welke werd beheerd door de Stichting Mechanische Registratie en Administratie (SMRA). Dit systeem verzorgde de landelijke administratie en bevatte de leden voor alle kerken. Vanuit dit mainframesysteem ontvingen de lokale kerken periodiek gegevens (bijv. eens per jaar de statistieken) en het was ook mogelijk om zelf bepaalde gegevens (selecties, kerkbalans, etc.) op te vragen. De gegevens ontving je altijd op papier.

Gemeenten wilden in het PC tijdperk in toenemende mate zelf de administratie kunnen bijhouden in een eigen database. Toen zijn ze begonnen met een nieuw, lokaal pakket dat de naam “Baruch” kreeg (naar de schrijver van het Bijbelboek Jeremia). Deze applicatie was gebouwd op basis van MS Access met Visual Basic schermpjes (macro’s) en communiceerde met de mainframe. Deze communicatie verliep via een ‘uitwisselbestand’ waarmee de gegevens bijgewerkt werden vanuit de mainframe of andersom. Als meerdere mensen binnen een

gemeente gebruikmaakten van het pakket werd het bestand gekopieerd en onderling uitgewisseld. Iedereen moest dus ook onderling synchroniseren en bestanden raakten corrupt. Dit was een ellende.
Daarna kwam het commerciële pakket “Scipio” op de markt. Dit pakket had oorspronkelijk een sterke verbandschap met Baruch, maar werd verder doorontwikkeld en daarmee kwam het ook verder af te staan van het oorspronkelijke concept en kerkordelijke inbedding. Dit pakket werd constant aangepast naar aanleiding van feedback van de lokale kerken (de gebruikers). Dit was lastig omdat het voor minder eenheid zorgde tussen de applicaties van verschillende gemeenten. Toen het pakket nog door de SMRA werd beheerd voldeed het pakket namelijk volledig aan de spelregels van de kerk. Nu het pakket commercieel was ingezet wilde niet iedere gemeente zich volledig conformeren aan deze spelregels en ontstonden er lokale afwijkingen in de applicatie. Gemeenten die (bijv. door de lokale omstandigheden) moeite hadden met het conformeren aan de standaardapplicatie waren erg gecharmeerd van de nieuwe oplossing.

Toen de kerken fuseerden en de PKN ontstond was er een mainframe met het pakket Baruch, de spin-off hiervan Scipio (dat nog steeds bestaat) en een paar exoten, andere pakketjes die sommige gemeenten zelf in elkaar geknutseld hadden. Het enige pakket dat synchroniseerde met het mainframe was het Baruch-pakket. Omdat de kerk fuseerde moest het systeem grondig worden herzien om aan te sluiten bij de nieuwe kerk en kerkorde; dat werd als een goede gelegenheid gezien om de technische structuur ook te herzien en te kiezen voor een web-gebaseerde oplossing. De mening van de gefuseerde kerk was dat het synchroniseren erg uit de tijd is. Een belangrijk principe voor de komende jaren is dat we de lokale gemeenten willen ontzorgen. Daarom wilden we af van het oude pakket en over naar een online centraal webbased systeem, zodat het enige dat een gemeente nodig heeft een webbrowser is. Zodoende zijn ze begonnen met een nieuw project dat de mainframe vervangt. Een belangrijk uitgangspunt was ook dat alle ouderlingen niet meer met kaartenbakken hoeven te werken. De ouderlingen gebruiken de gegevens die de ledenadministrateur bijhoudt. Bijna elke gemeente heeft een ledenadministrateur die de gegevens van de kerkleden bijhoudt. In de oude situatie kregen de ouderlingen dus papieren extracten voor hun deel van de gemeente in de vorm van een kaartenbak. In de nieuwe situatie is de kaartenbak digitaal. Iedereen die het voor zijn kerkelijke werk nodig heeft moest toegang krijgen tot de gegevens, maar wel een afgekaderd deel van de gegevens (bijv. bij wijkverantwoordelijkheid alleen de ledengegevens over die specifieke wijk en blijven financiële gegevens verborgen). Ook werd het als een handige toevoeging gezien als bijgehouden kan worden wanneer er een bezoek is gebracht aan een lid (als bijv. de dominee een maand geleden nog geweest is hoeft de ouderling niet perse deze week langs te gaan). Het was dus geen puur administratieve tool meer. “Het moet ook heel duidelijk een stuk gereedschap worden voor de gemeente, waar de kerkenraad mee aan het werk kan”. Event-driven marketing: voorstellen om op bezoek te gaan als er een baby is geboren.

De traditionele applicatie Baruch werd zo goed als niet beheerd. De mainframebeheerders (SMRA) brachten weleens een patch uit. Er kon dan een bestand worden gedownload of een CD opgevraagd worden. Het was puur een macro-applicatie, dus je kreeg dan gewoon nieuwe Visual Basic code. De database kon je zelf backuppen en restoren, dus dat deed de plaatselijke gemeente. Als er een PC gecrached was had je pech, maar je kon dan wel een nieuwe dump krijgen vanuit de mainframe. Belangrijk is dan natuurlijk wel dat je de data op de mainframe zelf goed bijgehouden had. De mainframe werd door de SMRA beheerd, door een man of 60 (voor beheer, maken lijsten voor gemeenten, ondersteunen gemeenten, kerkbalans draaien, drukwerk verzorgen). De lokale gemeente beheerde de applicatie. Er was wel een helpdesk die je kon helpen als je in de knoei kwam met het pakket, maar je moest het allemaal zelf doen.
Processen als configuratiebeheer en veranderbeheer of raamwerken als ITIL waren in deze applicatie helemaal niet aan de orde. De organisatie was niet erg professioneel. Iedereen deed wat goed was in zijn eigen ogen. Ze deden allemaal hun best, maar het was niet goed geformaliseerd. Er was wel redelijk veel gedocumenteerd, wat dat betreft was het een redelijk ambtelijke organisatie. Deze documentatie is allemaal afgevoerd en opgeruimd, dus niet meer beschikbaar.

De applicatie Leden Registratie Protestantse kerk (LRP)
We hebben drie awards gekregen voor de beveiliging van LRP.

In de eerste fase van het project hebben we 15.000 toegangssleutels gekocht, die zijn allemaal in gebruik en we hebben er nog eens 5.000 bijbesteld. Het aantal eindgebruikers zit dus nu ergens tussen de 15.000 en 20.000. De kerk heeft 2 miljoen leden die allemaal in de applicatie staan, plus dat de gemeente ook ‘administratieve’ adressen kan opnemen in het systeem, dus ik denk dat we ongeveer 2,5 miljoen adressen in het systeem hebben staan.

Het plan van aanpak is in een halfjaar geschreven. In Maart 2009 is het PvA goedgekeurd op de synode. Het projectplan is in dat jaar geschreven. In december 2009 kregen we groen licht voor bouwen, januari 2010 begonnen we met de bouw en een jaar later (januari 2011) is het systeem in gebruik genomen. De bouw was opgesplitst in een halfjaar bouw om een fundament te leggen en vanaf september 2010 zijn we pilots gaan draaien met 25 gemeenten die het pakket uitprobeerden en hebben we verbeteringen gedaan. Het is iteratief gebouwd middels 4 functionele elementen: kerkelijke stuur en organisatie (KSO), ledenadministratie, bijdrageadministratie, dienstverlening. Er zijn tot september 2010 dus vier reeksen van iteraties geweest; elk functionele blok werd in een periode van 6-10 weken gerealiseerd, waarbij er wekelijkse iteratieoverleggen waren. In totaal zijn er dus minimaal 4 * 6 = 24 iteraties geweest.
Het primaire doel van de applicatie is kerkewerk mogelijk maken. Daarnaast het leden in de gaten houden in de positieve zin van het woord. Hèt moment wanneer mensen namelijk van de kerk afhaken is wanneer ze verhuizen, dat is internationaal ook zo. Het is dus belangrijk gelijk iets van de kerk te horen als je verhuist bent. Voor de kerk is het belangrijk om zo snel mogelijk contact te leggen. De nieuwe gemeente krijgt in het systeem een bericht als mensen naar de gemeente toe verhuizen. Een ander doel van de applicatie is het benaderen van de leden voor fondsenwerving. Daarnaast is voor zowel de landelijke kerk als plaatselijke gemeenten een bijkomend doel het bijhouden van statistieken om het beleid daarnaar aan te passen (bijv. wanneer te zien is dat jongelui afhaken of iedereen in het dorp naar de buurgemeente gaat).

Een maat als ‘functiepunten’ is eigenlijk niet meer van toepassing bij het ontwerp van een applicatie wanneer je deze iteratief bouwt.
Dat het in Mendix zo makkelijk is om dingen toe te voegen is ook gelijk het nadeel. Hierdoor krijg je namelijk al heel gauw: “oh als je toch hiermee bezig bent, kun je dan ook daar even een veldje toevoegen”. We hebben dus heel sterk de gebruikers gevraagd welke functionaliteit ze moeten hebben en niet welke functionaliteit ze graag willen hebben. Duidelijk onderscheid tussen Must have en Nice to have is belangrijk. Het is hierdoor dat we het systeem binnen de tijd en binnen budget hebben kunnen opleveren.

De plaatselijke gemeente heeft nu niets meer met beheer te maken. Geen backups, geen software updaten, geen PC‘s updaten. Het enige nadeel is dat gemeenten dus ook niet een eigen backup kunnen maken alvorens risicovolle handelingen uit te voeren op het systeem, zodat ze een backup zouden kunnen terugzetten als het misgaat. Hiervoor hebben we in het begin van het nieuwe systeem wel eens een gemeente aan de telefoon gehad. Ze vroegen of wij “even een backup van het systeem konden terugzetten”. Dat ging dus niet. Een gemeentelid is namelijk niet meer expliciet verbonden aan één gemeente maar kan bij meerdere gemeenten geregistreerd zijn. Om het probleem op te lossen geven we de gemeenten een kopie van de productieomgeving, dit noem ik ‘de zandbak’. Hier kunnen ze alles op uitproberen en deze omgeving wordt eens per maand ververst met actuele gegevens (halverwege de maand). Als ze deze omgeving vernielen moeten ze maximaal 4 weken wachten en dan is alles weer goed. Dit is een oplossing voor het feit dat mensen graag iets willen kunnen uitproberen.

Mendix heeft geen documentatie. Ze zeggen: “het systeem is de documentatie”. De ontwikkelaar kan de ‘flow’ aflezen uit het model, maar voor de duidelijkheid moet de ontwikkelaar in eerste instantie wel de discipline hebben om erbij te zetten waarom hij iets op die manier doet. De applicatiebeheerder of ontwikkelaar kan het model ‘lezen’, maar wij hebben geen handboek dat beschrijft hoe de software in elkaar zit. “Als een gemeente vraagt: “hoe werkt jullie software eigenlijk” kan ik geen boek geven”. Onze applicatiebeheerder heeft wel een dikke handleiding gemaakt met uitleg over lokale beheertaken.
We hebben expres aangegeven bij Mendix dat we (rechts bovenin) een versienummer in beeld willen hebben staan om in de handleiding te kunnen refereren naar een bepaalde versie (bijv. “dit is opgelost in die versie”). Onze ervaring met de oude applicatie was dat we op een gegeven moment meerdere omgevingen hadden waarvan we niet wisten welke versie op welke omgeving stond. Dit versienummer is van belang als wij klachten binnenkrijgen uit het land. Als iemand vorige week een probleem tegenkwam wil je kunnen analyseren of dat inmiddels al is opgelost.
Het voordeel van de waterval-methode is dat alles op papier staat: functioneel ontwerp, technisch ontwerp. Eventueel moet je na de bouw dan wel nog de documentatie aanpassen, maar in principe heb je het al op papier staan op het moment dat je aan het bouwen gaat. Mendix werkt zo niet.

Management
Toen vorig jaar Mendix van boord van het project stapte zeiden ze dat we een partner moesten zoeken voor het onderhoud. We hebben toen een aantal partijen uitgenodigd om een aanbieding te toen (dat was rond

oktober 2011), maar het punt was dat geen van de partners van Mendix voldoende ervaring had. Ze hadden allemaal wel kleine applicaties gebouwd, maar niemand had zo'n grote applicatie gebouwd als LRP. Daarnaast was het voor ons ook van belang dat de partij kerkkennis heeft, omdat hij wel moet begrijpen hoe onze kerk in elkaar zit (met ouderlingen, predikanten, ledenadministrateurs, landelijke organisatie, etc.). Door beide punten konden we niet echt een geschikte partij vinden. De oude mainframeclub had het beheer van LRP overgenomen, maar daar zaten ook nog mensen in die eigenlijk overbodig waren geworden (ontwikkelaars bijv.). We hebben toen besloten om het applicatiebeheer zelf te gaan doen, samen met één voormalig ontwikkelaar van de SMRA en één van de mensen van Mendix. De persoon van Mendix mochten we een jaar lang binnen boord houden voor kennisoverdracht e.d., hij was zeg maar de whizzkid van het project. Hij besloot vorig jaar om voor zichzelf te beginnen en nu loopt deze man nog twee dagen per week bij ons rond. Wij doen dus met 2,4 fte ons eigen functioneel onderhoud. Het LRP team is een stuk groter dan dat en doet bovendien: ondersteuning gemeenten, fusies verzorgen, gemeenteadviseurs, orderadministratie, telefonische helpdesk. De helpdesk helpt de gebruikers. Hier komen gebruikersfouten binnen, maar ook data corrigeren die de gemeente zelf heeft foutgemaakt, door een softwarematige fout of door foute data in het systeem niet kloppen. In de laatste gevallen doen ze een beroep op onze beheerders. Deze beheerders zijn de 2e lijn. Repareren van fouten/bugs doen wij in ons eigen team. We hebben redelijk veel openstaan op dit moment.

Voor het bouwen van versie 2.1 zijn we een partnerschap aangegaan met een Mendix partner die de grote bouwprojecten voor ons uitvoert. Het uitbreiden van functionaliteit, nieuwe functionaliteit, optimaliseren besteden we dus uit. Kortom: het dagelijks onderhoud/ondersteuning/reparatie doen we zelf, maar bouw besteden we uit.
Vanuit de servicedesk komen veel wensen en bugs binnen. Dit voeren wij in in het pakket 'Sprintr'. We hebben een gebruikersplatform/belangengroep GPL (GebruikersPlatform LRP) opgezet die binnen de gemeentes polst wat de gemeenten belangrijk vinden. Voor de bouw hebben we onze eigen prioriteringsstelling, de MOZES-lijst; MOeten/Zou moeten/Eventueel/Stellen we uit. We delen alles in volgens deze prioritering en we hebben tegen de bouwer gezegd: "alles wat MO is moet je bouwen en als je nog een Zou moeten kunt meepakken moet je die ook meepakken. De eventueeltjes stellen we uit en dit is het budget". We zijn uiteindelijk over het budget gegaan omdat we daar niet goed over nagedacht hebben, maar ik kon mijn gebruikers wel tevredenstellen omdat alles wat Moeten/Zou moeten is gebouwd is. We hebben hiervan geleerd dat het te gemakkelijk is gegaan, te chaotisch en de GPL had het idee dat ze niet volledig gehoord waren. We doen dit nu anders. We hebben een procedure afgesproken dat wij geen wensen meer verzamelen. Wij verzamelen alleen nog maar bugs/verbetervoorstellen/reparaties in Sprintr. GPL verzamelt alle wensen. Alle wensen die binnenkomen bij de helpdesk gaan door naar de GPL en zij gaan toetsen bij hun achterban (de gemeenten intern én onderling) wat belangrijk is. Deze wensen beschrijven ze via een standaard sjabloon, goed omschreven en geprioriteerd. Wij toetsen daarna in hoeverre deze wensen technisch en functioneel verantwoord zijn (o.a. architectuur en performance) en vragen aan de bouwer hoeveel het kost. Er is dan dus een geprioriteerde wensenlijst waar wensen op staan met hun kostenplaats en doorlooptijd. Dat gaat terug naar het 'releaseteam' (= het GPL samen met de functioneel eigenaar van het systeem en ICT). Dit team bepaalt vervolgens welke wensen wel en niet in de volgende release worden meegenomen. Deze gaan vervolgens naar de bouwer die ze voor de eerdergenoemde fixed price bouwt. Hij gaat niet over de prijs heen en als de tijd zelfs meevalt geven we hem nog een 'reservelijstje'. Als de kosten erg tegenvallen schrappen we één of meerdere wensen. Tijdens de bouw komt de indiener van de wens weer aan boord om in een iteratief proces te bepalen of zijn wens correct is gebouwd.
Het servicedesk proces: de telefonisten beantwoorden de calls en er zit iemand die heel veel kennis van zaken heeft die bepaalt of het een gebruikersprobleem is of een probleem dat wij kunnen oplossen of een dataprobleem dat wij kunnen repareren. Deze persoon filtert de calls die naar de 2e lijn (Applicatiebeheer) doorgestuurd worden. De criteria en het proces hiervan zijn afgesproken. Als een issue naar de 2e lijn doorgaat komt het op de 'Sprinterlijst', wordt het getoetst en gefilterd en pakken wij het op in volgorde van prioriteit. Prioriteit heeft te maken met impact (hoeveel gemeentes treft het) en de ernst van de fout (bijv. als veel data omvalt of als het blokkerend is voor belangrijk werk binnen een gemeente).

Verschil tussen beheer van een SaaS applicatie en een traditionele applicatie is dat ik geen distributie hoef te doen; erg handig; ik weet zeker dat iedereen dezelfde versie heeft. Geen clients opwaarderen/installeren, dus je hebt er grip op. Het nadeel is echter wel dat gelijk alles platligt als je iets fout doet. Dit is één van de belangrijkste punten. De gebruikers hoeven ook nergens meer mee te klungelen, maar een nadeel hiervan is dat niet alle browsers gelijk zijn en oudere browserversies nog wel eens onverwachte situaties kunnen opleveren. Voor sommige mensen is het erg lastig om een browser te updaten. Mendix garandeert IE, maar IE6

was niet vooruit te branden. Google Chrome werkte in het begin ook lastig, maar werkt nu wel. We hebben de gebruikers aangeraden om Firefox te installeren. Toen wij begonnen was Firefox bovendien 40% sneller. Inmiddels heeft Microsoft wel een inhaalslag gemaakt. In ieder geval, je merkt dan pas hoe vervelend het is dat mensen software moeten installeren.
Een ander punt is dat het heel moeilijk is om een bepaalde groep op een pilot te zetten voordat we het uitrollen. Het is óf testen óf productie en de productieomgeving is hetzelfde voor iedereen. Je hebt dus nooit de mogelijkheid om iets uit te proberen, anders dan in de testomgeving. Maar de testomgeving is ook gelijk een testomgeving dus werk dat mensen daarin doen wordt niet bijgewerkt in de productieomgeving. Het is niet mogelijk met een klein groepje een nieuwe versie uit te proberen voordat iedereen het krijgt. Als je dan bijv. een performancefout hebt geïntroduceerd gaat het gelijk goed mis. Dus: gefaseerde invoer gaat niet.

Als projectmethodiek gebruiken we PRINCE2. Ons management is niet perse ingericht op een bepaalde standaard als ITIL. Maar ITIL is gewoon gebaseerd op gezond verstand, dus we hebben het proces ingericht op basis van wat ons verstandig leek. Ook dit maakt een soort iteratieve ontwikkeling door. Zo hebben we dit jaar bijvoorbeeld gemerkt dat we het niet goed genoeg hadden geregeld; te los, waardoor we over het budget heen gingen. Wat betreft configuration management heb ik de software op de server, die moet goed onder controle zijn (test en productieomgeving) en aan de gebruikerskant alleen de browserversie. Dat is niet zo spannend, dus daar hoef je ook geen grote dingen voor in te regelen. We hebben wel met Mendix goede afspraken gemaakt over het beheer van de server, ik wil bijvoorbeeld niet dat zij zomaar even een nieuwe Linuxversie naar binnen rijden. Op het platform hebben we dus wel configuration management, maar op de applicatie weinig. De applicatie is in principe één blok dat als één geheel wordt gezien voor beheer; het werkt niet met losse servicecomponenten. Ik heb begrepen dat de nieuwe (toekomstige) versie van het Mendixplatform wel met modules/plugins kan werken, maar op dit moment is het één geheel. Incident management doen we via de servicedesk. Het belangrijkste voor wat betreft change management voor onze applicatie is dat we goed testen, omdat je dus niet met een pilotgroep kunt werken. Ook communicatie naar gebruikers over changes is belangrijk, bijvoorbeeld door middel van een pop-up. Meestal doen we eind van de dag of ’s morgensvroeg een release, dat zijn de momenten dat de kerkelijke mensen het minst aan het systeem zitten. Dit kunnen we niet ’s avonds doen, want dan zijn als die vrijwilligers juist aan het werk. Rond etenstijd ’s avonds is een goed moment voor een reboot en maandagochtend voor een grote release.
Binnen de beheerprocessen spelen de volgende actoren een rol: de systeemeigenaar (die budgethouder is) is de manager van de afdeling waar het LRP Team ook onder valt, Hoofd ICT, Applicatiebeheerders, teamleider LRP team en servicedesk, GPL. Dit is het releaseteam.

Het technische staat bij Mendix: serverplatform, database (POST SQL). Zij regelen de virtuele indeling van het platform; cluster met virtuele servers. Mendix host de applicatie, wel op onze eigen hardware bij XS4ALL in Amsterdam, maar Mendix doet het technisch beheer ervan (Linux en dat soort dingen; onze beheerders zitten in de Microsoft-hoek dus hebben daar ook geen kaas van gegeten). Mendix is dus verantwoordelijk voor de performance van het systeem. We hebben een uitwijkomgeving, die stond eerder ook op onze eigen hardware maar die hardware is aan de productiehardware toegevoegd en nu is er een uitwijk via Mendix, as-a-service afgenomen. Onze data wordt dus gedupliceerd naar het Mendix rekencenter en in geval van een calamiteit kunnen we op hun servercapaciteit draaien.
De bouw van de applicatie wordt beheerd door XS2OS (dit is onze partner uit Barneveld). Functioneel beheer en klein onderhoud doen wij helemaal zelf.

Wij hebben een formeel ondertekend SLA met Mendix over het technisch beheer. Dit is gewoon de standaard SLA die Mendix afsluit met haar partners. Wij zijn onze eigen partner die onze functionele fouten oplost. We hebben geen SLA met de bouwer, want dat gebeurt op projectmatige basis; wordt alleen maar ingeschakeld bij een bouwproject. We hebben geen formele SLA met de gemeenten. We communiceren wel naar ze dat we 6x10 (ma t/m za, 10 uur per dag) ondersteuning garanderen. Dit hebben we ook met Mendix geregeld. Als ’s nachts om 3 uur het systeem omvalt wordt de telefoon niet opgenomen. Dit is een bewuste keuze geweest van de kerk. We hebben destijds becijferd wat het kost om 7x24 uur support te leveren en het risico weegt niet op tegen de kosten hiervan. “Het is maar een kaartenbak, de kerk gaat niet failliet”. In het geval van bijvoorbeeld een internationale bank of een webwinkel is de impact veel groter als het systeem eruit ligt. Het is een kosten/baten afweging geweest. De dienstverlening gaat niet verder dan 6x10 uur.
Als wij wijzigingen willen op het platform van Mendix staan wij ook gewoon in de rij als één van de klanten met een wens. In de tijd dat LRP gebouwd werd was dat makkelijker, want toen liep de technisch directeur van Mendix hier rond die zijn eigen afdeling inschakelde om de nodige functionaliteit te bouwen. Er is tijdens de

bouw van ons systeem redelijk veel specifiek voor ons gebouwd in het Mendixplatform; er zit schijnbaar ook redelijk veel in Mendix dat erin is gekomen omdat de kerk dat nodig had.
We zijn net gemigreerd naar Mendix v3.3. De v3.0 was specifiek voor ons gemaakt met patches die wij nodig hadden. Inmiddels zijn deze patches structureel onderdeel geworden van het systeem vanaf v3.1. Dit was tijdens de bouw, maar ik merk wel dat we nu ook gewoon één van de klanten zijn en we staan gewoon in de rij als er iets is. De prioriteit die Mendix eraan geeft bepaalt of ze er uiteindelijk ook iets mee doen.
In het beheer zijn wij vrij en onafhankelijk, maar ik ben hierin wel afhankelijk van de technische ondersteuning van Mendix, bijv. als ik er een database bij wil. Dit moet ik dan bij de technisch beheerclub van Mendix aanvragen en als de hardware vol is dan is de hardware vol.
Beheerprocessen die geautomatiseerd zijn: Mendix bewaakt het platform met monitoringtools waarin wij mogen meekijken. De applicatiebeheerder krijgt een sms/mail als het systeem plat gaat. Daarnaast draaien er 's nachts allerlei correctie- en analysescripts die bijv. data repareren waar iets fout is gegaan. Een voorbeeld hiervan is het herstellen van de leeftijd van een persoon die vandaag overlijdt, morgen jarig zou zijn geweest maar overmorgen pas in het systeem geregistreerd wordt als zijnde overleden. Deze leeftijd wordt dan rechtgezet.
Voor wat betreft rapportage heeft iedere gebruiker een 'meldingenscherm', als hij zich aanmeldt in het systeem ziet hij gelijk berichten als "er is een gezin bij je in de wijk komen wonen" of "die is overleden" of "er is een baby geboren". Ze noemen dit event-driven marketing, maar het is gewoon een statusoverzicht van de events die zich hebben voorgedaan rond hun mensen.
Wij zelf kunnen bepaalde analyses draaien op het systeem, we hebben rapporten/overzichten. Dit kan Arno je wel laten zien. We hebben ook een statistiekenmodule die telkens eind van de maand draait en dan allerlei tellingen door het systeem heen verricht en daarna grafiekjes geeft over het verloop van bijv. ledental/dopelingen. Het wordt dus maandelijks geteld en deze gegevens worden dan gecombineerd met andere gegevens om trends te produceren.

Arno Rog is nu anderhalf jaar aan boord als applicatiemanager LRP en kan je meer gedetailleerd het systeem laten zien; rollen, beveiliging, monitoringtools, etc.

Het SaaS management framework
Processen waar je in de praktijk wat mee te maken hebt zoals wijzigingenbeheer, incidentbeheer, configuratiebeheer concentreer je je meestal het meest op. Die regel je dan in.

Service catalog management: onderdeel van de service catalog is het uitwijkplatform dat Mendix as-a-service biedt én het beheer dat Mendix voor ons regelt. Dit is de dienstverlening die ik afneem van Mendix. Als je geen eigen cloudomgeving hebt kan Mendix ook geen uitwijkmogelijkheid bieden als dienst. Je krijgt dan andere soort contracten.

Contract/subcontractor management: Mendix heeft geregeld dat onze apparatuur bij XS4ALL staat. XS4ALL is dus weer een provider voor de internetlijn, rack, UPS en alles wat erbij hoort.

Change evaluation: je kunt nooit met 1.000 mensen tegelijk testen. Door afhankelijkheden krijg je soms een fout in het systeem die komt doordat mensen dingen tegelijk aan het doen zijn. Dit zijn dan dingen die je pas later in de praktijk tegenkomt.

Software management: dit heb je feitelijk bijna niet meer. Je hebt maar één versie van de software op een server staan die geldt voor iedereen. Ik hoef niet te managen dat mijn klanten allemaal dezelfde versie hebben; geen distributie. Ik moet natuurlijk wel zorgen dat de configuratie van mijn testomgeving één-op-één wordt overgebracht op productie. Je hebt dus wel nog distributie binnen je eigen platform, maar dat stelt eigenlijk helemaal niets meer voor. Daarom is de naam 'software management' toepasselijker dan de oude naam 'software distribution', dat laatste is helemaal niet meer aan de orde.

Configuration management: infrastructuur/hardware management is belangrijk voor het Mendixplatform. Voor de eindgebruiker is belangrijk dat hij een goede browserversie gebruikt, maar service components zijn bij LRP niet aan de orde. Configuration management is minder belangrijk geworden dan dat het was. Het hele configuratiegebeuren is gewoon een stuk makkelijker geworden met SaaS.

Knowledge management: LRP netwerk met handleidingen, forum, GPL forum. Sprintr is één grote knowledge database. Hierin staat geregistreerd welke fouten er gevonden zijn en hoe deze gerepareerd worden. Fouten kun je hierin terugzoeken. Het is een belangrijke tool voor ons.

Access management: wij autoriseren de zogenaamde lokaal beheerder in een gemeente en de rest bepalen ze binnen de gemeente zelf. Als een gemeente 15 sleutels bestelt geven wij de eerste sleutel 'lokaal beheerder'-autorisaties en de 14 andere sleutels hangen we aan die gemeente. Plaatselijk wordt dan in de gemeente bepaald wie welke sleutel krijgt. We hebben hierin delegation of control (zoals ze dit noemen in Active Directory) toegepast. De dienstenorganisatiemedewerkers hebben sleutels met rollen die redelijk veel mogen.

Event management: de monitoringtool die er is rapporteert naar de beheerders als er wat spannends speelt. Onze applicatie maakt geen gebruik van event-dingen. Als landelijke organisatie doen we wel een aantal dingen voor de gemeenten en daar zit wel event triggering op, bijv: wij mogen SILA mutaties verwerken. Wij krijgen de mutaties van de GBA en op het verwerken van die SILA zit wel signalering. Als hierin iets fout gaat moeten de mensen bij de centrale dienstenorganisatie actie ondernemen.

Service lifecycle management: regulier onderhoud (patches) en projectmatige bouw van nieuwe versies. Verder: issues komen binnen bij de servicedesk, worden gefilterd en gaan door naar de 2^{e} lijn waar ze in een sprint worden opgenomen.

Use support: is wat de servicedesk doet; mensen helpen en fouten aannemen, wensen aannemen en doorspelen naar request fulfillment die het wensenlijstje in de gaten houdt.

Problem management: is altijd gerelateerd aan incidenten. Als je herhalende incidenten hebt dan weet je dat je een structureel probleem hebt. Het onderscheid tussen incidenten en problems wordt bij ons niet specifiek geregistreerd. Wij leggen de incidenten vast in Sprintr en als herkend wordt dat we een incident al eens eerder hebben gehad gaat hij wat hoger in de prioriteitenlijst als structureel probleem dat snel moet worden opgepakt.

ICT operations management: Dit is in principe iets wat wij bij Mendix hebben belegd. Zij houden in de gaten of het geheugen niet stukloopt en de harde schijf niet volloopt. Mendix bewaakt het en laat het ons weten als er bijv. een processor of schijf bij moet komen.

ICT operations control: Monitoren/backups/job scheduling gebeurt door Mendix.

We hebben vorig jaar/ begin dit jaar een hoop moeten bijstellen in de beheerprocessen dat niet goed ging. We hebben naar aanleiding hiervan de regels wat strakker ingesteld. We zijn nu op een punt dat de processen redelijk strak zijn ingeregeld, redelijk reproduceerbaar en het meeste staat op papier, maar niet alle procedures zijn exact omschreven.

INTERVIEW REPORT A-I2

Interviewee position: Application manager LRP
Interview date: 19-10-2012
Interview duration: 149 minutes

Applicatiebeheer LRP
PKN is wat betreft SaaS een buitenbeentje omdat LRP weliswaar extern gehost wordt, maar niet zozeer een traditionele Mendix SaaS applicatie is. We hebben eigen hardware, die wordt beheerd door Mendix maar deployen in de cloud als in "met één druk op de knop je model geautomatiseerd in een serveromgeving brengen en daar dan ook starten" is bij ons niet aan de orde. Dit heeft te maken met de omvang van LRP en de benodigde resources. De Mendixcloud heeft daar gewoon onvoldoende capaciteit voor. We hebben dus eigenlijk onze eigen private cloud, ook met het oog op de beveiliging. Omdat we niet met één druk op de knop kunnen deployen in de Mendixcloud hebben we iemand aan boord met Linuxkennis. We kunnen letterlijk op de servers de bestanden bekijken, backups maken en restoren op een andere omgeving.

Feedback kan intern (dienstencentrum) of via de helpdesk extern (eindgebruikers) zijn ontvangen. We maken geen gebruik van de feedback-knop die Mendixapplicaties vaak aan boord hebben omdat we ons niet kunnen permitteren dat 10.000 meer of minder gefrustreerde personen van zo'n knop gebruik gaan maken. De helpdesk filtert in eerste instantie en als die er niet uitkomen wordt het als feedback ingeschoten en gaat het naar ons applicatiebeheerders als 2e lijn. Wij beoordelen dan nogmaals of het klopt en de helpdesk niet iets geks over het hoofd heeft gezien. Bijv. persoon A heeft een bankrekening weggegooid en persoon B gaat een acceptgiro draaien. Dan zegt het systeem natuurlijk: "dat gaat niet lukken" en dat kan het heel elegant doen of het vliegt er knalhard uit met onbekende fout. In zo'n situatie koppelen we de feedback terug naar de helpdesk dat het rekeningnummer ontbrak.

"Ik heet officieel applicatiebeheerder, maar je kunt het ook wel haarlemmerolie noemen." (Wikipedia: Haarlemmerolie is een middel dat uit de zeventiende eeuw stamt. Het wordt al eeuwenlang aangeprezen omdat het tegen alle kwalen zou helpen). Nadat de eerste screening heeft plaatsgevonden beoordeel ik of het een procedurele fout is geweest of dat het programma niet helder genoeg is geweest of dat het een technische afwijking is geweest en LRP daadwerkelijk zijn werk niet goed heeft gedaan of was de data niet valide of is het een wens. In het geval van een technische afwijking zullen we sowieso een correctie moeten doorvoeren. In geval van een wens moet de aanvrager er nu even mee leren leven en gaan we kijken of we het voor de volgende sprint kunnen opvoeren.
Voorheen werden wensen altijd door onszelf beoordeeld naar eer en geweten, impact en rendement ervan. We moeten hier efficiënt mee omgaan omdat we niet alles kunnen oppakken. We hebben nu een nieuwe partij aan boord, gelieerd aan ons, met de naam GebruikersPlatform LRP (GPL). Dit is voor ons een klankbord en via hun krijgen wij ook feedback uit een gestructureerde omgeving. Het GPL bestaat uit werkgroepen, bijv. speciaal voor bijdragenadministratie, ledenadministratie, lokaal beheer, pastorale werkzaamheden. Het GPL verzamelt alle wensen. Bugs sturen ze door naar de dienstenorganisatie in Utrecht. Het GPL is een formele partner in het overleg waarin besloten wordt wat we met de applicatie gaan doen voor de volgende release. Het GPL verzamelt de wensen en geeft er een gewicht aan, op basis van de ernst die ervan in het land wordt ervaren en wij hebben hier naar te luisteren door dit te realiseren.
Als een gemeente via de helpdesk aan mij vraagt om bijvoorbeeld bestanden weg te gooien (omdat ik die autorisatie heb en zij niet) doe ik dat in de testomgeving 'de zandbak' voor ze, zodat ze eerst goed kunnen zien wat de gevolgen daarvan zijn. Als ze vervolgens concluderen dat dat is wat ze willen, wil ik dat graag formeel (schriftelijk) van ze horen en doe ik hetzelfde op de productieomgeving.

Applicatieontwikkeling LRP
Mendix gebruikt een teamserver (soort centrale repository) waar alle bestanden in staan. Iedereen die werkt aan LRP kan een kopie van het model naar zijn eigen PC toetrekken. Na wijzigingen test je het eerst zelf en als het goed is "commit" je de wijzigingen. Hiermee stuur je de wijzigingen terug naar de teamserver en haal je tegelijkertijd de wijzigingen van anderen op. Als dit conflicteert met andere wijzigingen van collega's biedt het systeem de mogelijkheid om de eigen wijziging of de wijziging van de ander te behouden of te kijken wat gewijzigd is en te komen tot een goede combinatie. Het is ieders verantwoordelijkheid om regelmatig te synchroniseren en dat doen we ook. Het model staat dus centraal en bewerken doen we lokaal. Als we met elkaar alles gerealiseerd hebben dat we in de huidige sprint wilden realiseren wordt een proefomgeving voorzien van de nieuwe versie van het programma. We testen in deze omgeving elkaars werk, we gaan uiteraard niet eigen vlees keuren. Het testen gebeurt door te kijken naar de oorspronkelijk ingebrachte feedback welke door ons is beoordeeld en omgezet naar een issue waarin we beschrijven wat we hebben geconstateerd. We hebben hierbij al gelijk kunnen aangeven in welke hoek het probleem naar onze verwachting zit. De feedback bestaat op dat moment uit: de probleemomschrijving en de beschrijving van wat de ontwikkelaar ermee gedaan heeft. We beoordelen dan of het klopt wat hij gedaan heeft en of het gedrag van de applicatie nu is zoals het gewenst is. Als dat goed is accepteren we het. Van een week ontwikkelwerk zijn we met 2 man 1 dag bezig om het door te testen. Als het in orde is keuren we de issues formeel goed en sluiten we de sprint. Meestal releasen we dan aan het einde van de dag het model naar de productieomgeving. Die dag geven we ook de eindgebruikers al een melding in beeld "Let op, heden middag vanwege gepland onderhoud zal LRP gedurende ongeveer een kwartier uit de lucht zijn". Het nieuwe model dat dus getest is sturen we dan op naar de productieserver, daar pakken we het uit/geven we de instructie "start dit model". Als je geluk hebt is hij daar heel snel mee klaar en als er wijzigingen in de database plaats moeten vinden is hij er even mee bezig. Als hij klaar is geeft het systeem aan dat het weer succesvol gestart is. We loggen dan minimaal even in om te zien of het systeem respons geeft, we hogen het versienummer op en we halen de melding voor de eindgebruikers weg en de technisch geïnteresseerde mensen krijgen dan via het LRP netwerk

(= ons communicatiekanaal) een patchlist waarin we de zaken voor specifiek intern gebruik binnen de dienstenorganisatie (bijv. t.b.v. de orderadministratie) buiten beschouwing laten, want eindgebruikers vragen zich anders alleen maar af wat dat voor gevolgen heeft voor hen.

SaaS management framework
Service Catalogue management: Mendix heeft net een mooie tool ontwikkeld hiervoor. Deze tool komt over een paar weken uit en helpt bij het bijhouden welke applicaties je hebt draaien + welke versie ervan + in welke fase van ontwikkeling zijn ze + draaien ze daadwerkelijk of moeten ze nog gedeployed worden. Deze applicatie van Mendix geeft al deze eigenschappen in een matrix weer. In onze cloud: welke applicaties heb je draaien, wie mogen erbij, wie werken eraan, wie mogen changes doorvoeren, enzovoort. Deze applicatie heet Enterprise .. manager o.i.d. en is webbased. Het idee hiervan is “hoe word je wijzer uit je app-jungle”. Hiermee heb je je servicecatalogus (app-catalogus dus) in beeld. De PKN heeft een hele hoop applicaties, maar voor mijn specifieke tak is deze catalogus redelijk eenduidig; we hebben er maar één.

Contract management/subcontractor management: wij hebben een bijzondere positie, want wij zijn deels zelf partner van Mendix. De aspecten waar wij geen kaas van hebben gegeten: technisch beheer, Linux beheer, serverbeheer, schijfbeheer, capaciteit, infrastructuur hebben we bij Mendix neergelegd en daar hebben we ook een SLA op met ze afgesloten. Deze SLA is op maat aangepast voor ons, want de beschikbaarheid is óf 24x7 óf beperkt zich tot 5 werkdagen en dat laatste is voor ons net iets te krap. Wij moeten eigenlijk rekening houden met een 6-daagse werkweek kun je zeggen. Op de zaterdag gebeurt er nog veel in LRP. Mendix zal geen subcontractor ten behoeve van ons inschakelen, want dat zijn we in principe zelf. We zijn zelf partner, maar we hebben zelf ook een partner die is gespecialiseerd op Mendix. Daarnaast huren wij intern één man in speciaal ten behoeve van zijn expertise en het feit dat hij grotendeels LRP ontworpen en gebouwd heeft: Herbert, heeft een eigen bedrijfje. Hij wikkelt de problemen af die wij nog niet gemodelleerd krijgen (vanwege de complexiteit of de ingrijpende consequenties op de database). Hij past dus óf het model aan, óf hij grijpt rechtstreeks in op de database. Ouderwets programmeren doet hij niet. Er is binnen Mendix niemand die zijn ervaring met LRP kan evenaren. Hij kan ook het Mendix-framework debuggen en is technologisch onderlegd om dan met Mendix R&D contact op te nemen en uit te leggen wat er niet goed gaat.
Mendix heeft bij de bouw van LRP hun tool steeds kunnen verbeteren en optimaliseren gebaseerd op onze feedback, want alles wat maar zwak was viel meteen door de mand bij zo’n grote applicatie. Nu staat het product veel volwassener in de markt, maar nog is men bij Mendix altijd wel erg geïnteresseerd hoe het bij de PKN functioneert om te kijken of het dan ook wel blijft werken. LRP is nu nog steeds de grootste Mendix-applicatie.
Komend jaar gaan wij over naar het nieuwe framework van mendix (v4), we zitten nu op v3. V3 is speciaal toendertijd voor ons gemaakt. We hebben een tijd de kat uit de boom gekeken, maar op een gegeven moment krijg je toch bericht dat v3 nog maar een jaar wordt ondersteund, dus we zullen een keer naar v4 over moeten gaan via een algemene v3.3. Voor het overgangsproces van v3.0 naar v3.3 nemen we 6 weken. Hiervoor hebben we alle processen in LRP nu geïdentificeerd en beschreven. Ik heb 20 testers die alleen maar bezig zijn om te kijken of het oude gedrag gelijk is aan het nieuwe gedrag. De testers zitten hier niet intern. Dit zijn in principe allemaal eindgebruikers of trainers of opleiders. Er is een groot verschil tussen doorgewinterde testers en mensen waarmee we de oorlog niet gaan winnen. De mensen die opleidingen en trainingen verzorgen weten natuurlijk precies hoe het moet werken. Het gemeenteproces testen we in 3 weken, daarna corrigeren we wat we gevonden hebben, maken we een nieuwe testomgeving en gaan we intern de volledige dienstverlening testen. Dat betekent voor 100 gemeenten álle brieven genereren, die gaan allemaal pro forma naar de drukker, maar de drukker drukt er maar eentje af. De hele keten van LRP bestelproces gaat via Microsoft CRM naar de gebruikelijke kanalen om het bij de printerbroker te krijgen en de printerbroker wijst een printer aan en print het. Het helemaal doortesten hebben we ook 3 weken de tijd voor (met tijd voor correcties), dan gaan we productie updaten met onze bevindingen en dan gaan we definitief productie draaien. Onze partner krijgt volgend jaar dan de opdracht: migreer het model van v3.3 naar v4, want daar zitten zóveel technologische en architectonische wijzigingen in (ook in het model) en een hoop handmatige oplossingen kunnen dan door het framework worden afgehandeld. Onze partner is Robert van den Ham van het bedrijf XS2OS. XS2OS is een firma in Barneveld die bestaat uit 3 of 4 Mendix-ontwikkelaars. Dat is onze formele partner. Als wij werk hebben uit te besteden doen we dat bij onze partner. Nadat onze partner het model van v3.3 naar v4 heeft omgezet gaan we daar een testomgeving voor inrichten. Onze eigen man zal zorgen dat de data zich correct gedraagt en dan gaan we misschien wel acht weken lang testen. Dit kun je niet echt iteratief testen, want het gaat gewoon om een final approve. Het model kan overal omgekukeld zijn.

Change management: dit begint bij het indienen van de change (een change request); iemand zegt: “dit werkt ons inziens niet goed, willen we graag gewijzigd hebben en we denken daarbij aan dit en dit en dit”. Formeel is het aan ons applicatiebeheerders zelf om te bedenken hoe we dat dan willen gaan vormgeven, letterlijk in: user interface, gewenste eindgebruikergedrag en gewenst gedrag in de applicatie tot en met databasewijzigingen aan toe. We gaan dit nu formaliseren. We willen van het GPL een geformaliseerde beschrijving hebben voordat we een change überhaupt in behandeling nemen, tot en met een soortement mockup (tekening hoe het eruit moet zien, mag kladje zijn). Applicatiebeheer maakt dan de afweging of het een kleine of spectaculaire aanpassing is en we wegen af wat het technisch risico is, impact op de database, hoe lang gaat het duren en wat is het rendement dat we ervan kunnen verwachten. Dit leggen we dan naast wat het GPL zegt. Als bijvoorbeeld 33 van de 2.000 gemeenten een probleem ervaren is het maar de vraag of de overige gemeenten er ook bij gebaat zijn. We houden hierbij ook een langetermijnvisie in de gaten. Wellicht biedt het bijvoorbeeld mobile-toepassingen in de toekomst. Er is dan een lager rendement op de korte termijn dat voor lief wordt genomen als het voor de toekomst interessant kan zijn. Anderzijds kun je, als je een bepaalde wens oppakt, ook verzanden in een woestijn van allemaal extra ad-hoc separate verlangens en wensen van mensen en dan moet je de keuze maken om zo’n wens niet op te pakken.
De aspecten die we meenemen in onze afweging wat we wel en wat we niet gaan bouwen zijn: gewenste gedrag vanuit de optiek van de eindgebruiker, weging ten aanzien van de impact op het model, impact op de database, impact op functioneren (technische dingen als bandbreedte), hoe zit het kerkordelijk (wettelijk), tijd, budget.

Change evaluation wordt gefaciliteerd door de Mendix tool. Als eenmaal besloten is een wijziging te doen kun je het namelijk razendsnel laten zien. Je hoeft geen complete evaluatiecommissie op te richten omdat je het aan de voorkant zo optimaal mogelijk beschrijft en de stakeholder betrekt bij het ontwikkelproces. Tijdens ontwikkeling door XS2OS hebben we wekelijks een iteratieoverleg, ontwikkelen ze deels hier in huis en kunnen we altijd meekijken op afstand. Als een bepaalde wens is ontwikkeld door XS2OS laten ze ons zien dat het werkt, waarna we de stakeholder (belanghebbende/indiener) erbij halen. De stakeholder geeft aan of het voldoet of dat er nog met andere aspecten rekening gehouden moet worden. Daarmee geven we de vrijbrief “dit is zoals we gedacht hadden, ga maar door”. Vanaf dat moment gaan we zo’n stuk gerealiseerde functionaliteit ook niet meer afschieten, dat is in het verleden teveel gebeurd. Nu hebben we ervoor gekozen de eis te stellen dat het voor aanvang voldoende formeel beschreven is en zo bouwen we het.
Change evaluation zit dus deels vóór change management en deels tijdens de ontwikkeling van de wens (namelijk na de eerste iteratie), niet meer gedurende het proces want dat kost teveel. Het hangt altijd samen met Change management, maar je kunt het wel zien als twee verschillende processen.

Software management: distributie is optimaal geregeld. Versiebeheer (software releases) wordt ideaal door de tool ondersteund, met dank aan de cloudomgeving van Mendix. De cloud is hierin een belangrijk aspect; het is de Mendixcloud, niet onze private cloud. De ontwikkeling gebeurt lokaal en daarna gaat het model naar de Mendixcloud (in de repository). De lokale ontwikkeling (en synchronisatie met de repository) duurt een week, want zo lang duurt de iteratie. Dan formaliseren we de status ervan (geven het een versienummer), alles wat we dan nog in onze werkvoorraad hebben zitten (= wat we niet gehaald hebben) volgens de SCRUM methodologie verplaatsen we terug naar de backlog of naar een volgende sprint. Het model dat dan in de cloud staat plaatsen we in een proefomgeving en dan gaan we de werking ervan testen (niet ons eigen werk, maar andermans werk), daar doen we een dag over. Als alle items goedbevonden zijn accepteren we de attributen in een tool van Mendix die dit ondersteunt. Deze tool bevat alle functionaliteit die we het attribuut “ready to test” hebben gegeven, die accepteren we dan. We deployen niet voordat alle punten die op “ready to test” staan geaccepteerd zijn of naar een volgende versie zijn verplaatst. Na het testen wordt de productieomgeving offline gehaald, het nieuwe model verplaatst naar de productieomgeving en wordt de productieomgeving weer online gebracht. Daarna draait de productieomgeving met de nieuwe functionaliteit. Versiebeheer is heel eenduidig gedefinieerd. Je kunt ook branches maken om grondiger iets uit te testen. Branches kunnen worden gemerged. We hebben maar één tool, de mendix modeler, in andere tools kunnen we niet werken. Er is altijd maar één versie van het model in gebruik, die staat in de repository en hebben we als ontwikkelaars. De wijzigingen die ontwikkelaars maken zijn maximaal een halve dag werk, daarna synchroniseren we met de server. Afspraak is maximaal een halve dag.

Configuration management: zoals mendix nu is, is er geen onderscheid in service components. Een Mendixapplicatie zal niet uit components bestaan die met verschillende versies zijn voorzien. Dit gaat overigens wel komen. Mendix wil graag in v5 (volgend jaar op z’n vroegst) modularisatie gaan toepassen; dat je

inderdaad een applicatie in modules kunt knippen, apart modules kunt bijwerken en ook modules onderling kunt laten communiceren volgens geformaliseerde methoden (vraag-antwoord concept). Communicatie kan met een eigen module, maar ook van andere partijen. Zo kun je met je module een algemene dienst aanbieden (zoals bijv. een straatnaam teruggeven op basis van postcode en huisnummer). Hiervoor heeft Mendix natuurlijk die Enterprise omgeving opgericht, want zij voorzien dat versiebeheer daarvan belangrijk wordt.
Op dit moment is de applicatie een afgerond geheel; alle modules en onderdelen zijn allemaal van dezelfde versie. Voor wat betreft configuration management is op dit moment de afstemming tussen software (die wij maken) en hardware (die Mendix beheert) belangrijk. Bijvoorbeeld als je statistieken bijhoudt moet je er rekening mee houden dat de database iedere maand gigantisch hard groeit. Dit is een gevolg van een softwarematige keuze, maar heeft gevolgen voor Mendix; zij moeten er qua configuratie rekening mee houden dat dit kan. Wij kunnen zien wat de performance is van de database op de servers van Mendix, qua geheugen, processor, schijf (bijv. om te controleren wat het effect is als we 10.000 brieven draaien). We kunnen de Java-runtime een beetje parameteriseren door geheugen anders toe te wijzen. Dit is voor de standaard Mendix klanten niet van toepassing, want die deployen rechtstreeks in de cloud en hebben met de techniek verder niets te maken.
Voor wat betreft configuration management op software niveau hebben wij eigenlijk twee hele grote items (database en applicatie) met een aantal eigenschappen die we bijhouden/onthouden. De onderliggende technische configuration management ligt bij Mendix. Als Mendix straks met losse modules gaat werken krijg je een strakker versiebeheer, want je moet dan steeds bedenken: “is de wijziging die we hebben doorgevoerd ook voor andere modules relevant en moeten die wellicht ook daar doorgevoerd worden”. Dat is nu helemaal niet van toepassing, zelfs als er maar één regeltje verandert moet de hele applicatie offline en opnieuw worden gestart. De hardware monitoring tool communiceert wel met ons. Op de hardware zit een zware monitoring. Elke ochtend ontvang ik diverse mails waarin de proefomgevingen aangeven dat ze in een kritieke belasting zitten, of de proefomgeving waarschuwt dat hij bijna vol zit door alle backups. Als de productieomgeving uitvalt en hij detecteert dat, krijg ik het meteen op mijn telefoon gemeld via sms alerts. Alle omgevingen worden volcontinu op een heleboel parameters gemonitord. Dit gaat binnen een paar seconden; als ik op afstand een groot bestand weggooi heb ik binnen 2 tellen op mijn telefoon een melding “de toestand is nu weer genormaliseerd”. We worden optimaal op de hoogte gehouden of het draait en of het goed draait. Bijvoorbeeld als in de Java-engine het proces van opruimen van tijdelijke bestanden een error heeft gehad en op een half been zit te hinkelen krijgen we steeds meldingen dat het systeem volloopt. We geven dan de melding dat het systeem vandaag een kwartier uit de lucht gaat wegens ongepland onderhoud en dan herstarten we het framework.

Knowledge management: wij registreren alle zaken die aangemeld worden, beoordeeld worden, geaccepteerd worden en realiseren in een online systeem dat ‘Sprintr’ wordt genoemd. Dat is de SCRUM tool zal ik maar zeggen. Als er iets geconstateerd wordt dat relevant is voor de collega’s zetten we het in dit systeem; wat hebben we gezien en wat hebben we eraan gedaan (technisch verwoord). We kunnen op deze manier alles terugzien op ieder gebied dat ooit gedaan is, vanaf de eerste fase van LRP. Ik had liever gehad dat dit wat meer mature en meer ontwikkeld was, als ik het vergelijk met andere pakketten bieden andere pakketten vaak iets meer. Maar het integreert naadloos met onze ontwikkeltool, dus ik ga er vanuit dat het uiteindelijk steeds meer uitgebreid en beter doorzoekbaar wordt. We leggen dus alles vast en daarnaast doen we aan versiebeheer en zijn we in enige mate in staat om na te gaan wat er is gewijzigd op een bepaald moment. De Mendixtool voorziet wel in documenteren, maar is hier niet optimaal in. De tool is erg grafisch ingesteld waardoor je bijna kunt “lezen” wat het systeem doet, maar soms wil je de overwegingen er graag bij hebben staan. Je kunt er wel comments bij zetten, maar het biedt niet de mogelijkheid om bijvoorbeeld de technische documentatie te bieden van de laatste sprint. Misschien is dat ook wel een beetje old school. Ik ben ook niet van het documenteren zó erg dat je niet meer aan werken toekomt, maar soms is het wel prettig om even structureel vast te leggen waarom we bepaalde dingen hebben gedaan. De mogelijkheden om dit formeel en gestructureerd vast te leggen zijn niet uitgebreid aanwezig. Het idee is om alles grafisch te doen. Dit red je vaak niet, omdat bijvoorbeeld onze verhuisflow afgedrukt een wand kan vullen, zonder de vele subprocessen. Het paradigma “dan lees je het toch” ga je daarom vaak niet redden in complexe dingen. Ik werk er nu een jaar aan, Jan (collega applicatiebeheerder) wat korter en allebei moeten wij nog vaak leren wat het gevolg is van bepaalde acties. Je kunt het wel lezen en het oogt visueel aantrekkelijk, maar een formele old school variant van het model is er niet. Dit wordt niet als blokkerend ervaren.
Conclusie: knowledge management is voor ons vooral de issues + oplossingen ervan (feedback). Zelfs van simpele vragen vanuit een gemeente worden issues gemaakt en die kun je dus teruglezen. Database queries

worden verzameld en gerubriceerd, om te voorkomen dat het wiel telkens opnieuw moet worden uitgevonden. De helpdesk doet los hiervan ook knowledge management; kennis verzamelen.

Access management: op eindgebruikerniveau gebruiken we sleutels. We hebben een 3-voudige wal: een geldige loginnaam (=geautoriseerd op een bepaald niveau, er is dus geen allesomvattende), wachtwoord en een digid code die elke 10 seconden wijzigt. Binnen de dienstenorganisatie zijn mensen met heel veel rechten, lokale gemeenten kunnen alleen bij hun eigen gemeente. Eén persoon is binnen de gemeente verantwoordelijk en kan andere personen binnen de gemeente aanwijzen die zich met de financiën of met de ledenadministratie kunnen bezighouden of ouderling/pastoraal werker. Mensen krijgen een klein eigen gebiedje toegewezen waar ze alles in mogen zien (bijv. een ouderling ziet zijn wijk, maar niet de financiële gegevens). Het is functioneel sterk gekaderd en autorisaties zijn gecompartimenteerd.
Wat betreft serververeiligheid: communicatie vindt plaats via een veilige https verbinding. De browser laat niets achter bij het afsluiten; alles wordt real-time gerenderd en er wordt niets aan caching gedaan; sessie sluiten betekent sessie weg. Persoonsgegevens kun je niet uit de cache lepelen, want die is er niet. Als ik bij mendix op de server wil moet ik me bij Mendix aanmelden, ik krijg dan een private key met 2-weg versleuteling en daarna kun je pas verbinding maken.
Voor eindgebruikers is architectureel geregeld, voor ontwikkelaars is het procedureel geregeld en gebaseerd op technologische belemmeringen als zijnde keys.

Event management: wij moeten er bijv. bij de ontwikkeling rekening mee houden dat het microsoft CRM pakket het moet kunnen accepteren als wij een ‘bericht’ veranderen. Er is nu een voorstel gedaan bij registreren in Sprintr al meer formeel aangeven op welke externe systemen (Microsoft CRM, drukker, externen via webservices) het invloed zal hebben en welke partijen eventueel betrokken moeten worden. Dit formeel vastleggen wordt nu belangrijker omdat het systeem dusdanig complex begint te worden.
Richting eindgebruikerskant communiceren we ook, bijv. in de wekelijkse nieuwsbrief, of de lokale gemeenten de registraties even goed willen controleren als de kerkbalans eraan komt.
Ik ga er vanuit dat als Mendix wijzigingen toepast (op bijv. de Linuxomgeving) dat wij daar ook van op de hoogte worden gesteld, ik ken niet alle ins- en outs van Linux dus ik ga er dan vanuit dat ze de wijziging hebben gedaan om iets te voorkomen. Ik zou het ook fijn vinden als zij zelf ingrijpen als ze zien dat het uit de hand dreigt te lopen of ons vragen wat we willen, die interactie moet gaan plaatsvinden.
Er is geen geautomatiseerd eventmanagement, maar op hardwareniveau is er wel sms alerting door de monitoring tool. Als processen niet goed lopen zal LRP dat meestal loggen in begrijpelijke taal voor ons met daaraan gekoppeld een java stacktrace zodat we kunnen kijken bij welke instructie het fout is gegaan. Als het systeem onderuit gaat zal het automatisch restarten. Er is geen intelligentie die bedenkt dat het ook via B kan, als het via A niet lukt. Als A niet lukt gaat hij niet verder en geeft daar een melding van.

Service lifecycle management: we doorlopen de cycle eigenlijk op tweeërlei vlak: wekelijkse sprints op basis van wat we teruggemeld krijgen waarbij het aspect ‘design’ niet aan de orde mag zijn, omdat we op basis van feedback in principe niet ontwikkelen, hooguit eens een vinkje of kolommetje erbij, maar geen hele nieuwe functionaliteit. We doen impact analyse, realisatie (binnen een week, bij voorkeur binnen een dagdeel moet het gerealiseerd zijn), deployment naar proefomgeving waar we andermans werk testen, indien gelukt implementeren we het. Dan is de documentatie en de realisatie ervan terug te vinden in Sprintr.
Het tweede vlak is bouwen van nieuwe functionaliteit, dan komt de partner (XS2OS) in beeld. We leveren ze een formele opgave van de te realiseren punten. Daarvoor filteren we het eerst (wel/niet bouwen) en bedenken we hoe we het ongeveer willen hebben; impactanalyse, rendement en beschikbaar budget. Design van de end-user kant lag voorheen bij ons, maar nu doen we dat in overleg met de eindgebruiker. Dan vertelt XS2OS hoe lang ze verwachten daarover te doen en hoeveel dat zal gaan kosten. Dat realiseren ze dan, testen het zelf, brengen het op de proefomgeving in de benen zodat wij het intern even kunnen testen. Als alles dan is opgeleverd staan heel veel zaken op “ready for testing”. We halen dan ons legertje aan testers binnenboord en we wijzen testers toe aan een specifiek onderdeel van het systeem op basis van interesse van de testers. De testers testen het, daarna kijken we of is opgeleverd wat we verwacht hadden en of het betrouwbaar is. Daarna vindt pas de implementatie plaats.
Het gebeurt dus wekelijks of in grote slagen (dat kan één of twee keer per jaar zijn). Voor komend voorjaar moet bijv. het SEPA verhaal gebeuren. Hiervoor maken we niet de analyse of het de moeite waard is om het te doen, want het moet gewoon (wetgeving).

Use support: is tweeledig; we hebben met 'het land' te maken (10.000 gebruikers), die worden door het serviceapparaat afgevangen. Daarnaast hebben we ook een interne afdeling waarvan het wel rechtstreeks bij applicatiebeheer komt te liggen, omdat we ervan uitgaan dat die mensen zelf van de hoed en de rand weten.

Request fulfillment: we maken een onderscheid tussen wat er binnenkomt en valide blijkt te zijn (bijv. een datareparatie; integriteit staat bovenaan bij zo'n groot systeem). Daarna maken we een onderscheid in 'wensen' en 'issues'. Wensen hebben we vroeger nog wel eens meegenomen, nu zijn we er wat formeler in omdat we het beter inzichtelijk willen krijgen en onze werkvoorraad best wel omvangrijk is en niet met de mate slinkt die we zouden willen. Zelfs wensen die gerealiseerd kunnen worden met een kleine aanpassing parkeren we even op de requestlijst omdat we er over een half jaar of een paar maanden wel eens anders tegenaan zouden kunnen kijken. Als het een issue is die gecorrigeerd moet worden prioriteren we het eerst en dan maken we de analyse of we het voor één gemeente doen of voor meerdere (kán de issue meerdere keren voorkomen en is het misschien ook al vaker voorgekomen). Het kan namelijk zijn dat de gemeente structureel iets fout doet. Voor de echte requests brengt het gebruikersplatform gradaties aan (impact op data, impact op het programma, impact op gebruikersgemak en het gemak/ongemak dat het land ervan ondervindt, hoe lang het al ter tafel ligt).

Problem management: mensen brullen gauw dat ze iets een "probleem" of "onacceptabel" vinden. Het kan zijn dat wij daar anders over denken en zeggen dat het niets met "omacceptabel" te maken heeft, maar dat het gewoon "by design" het beoogde gedrag is. We formuleren het dan als een wens. Als het echt een probleem is zijn we het daar allemaal gauw over eens, want dan kan een gemeente niet verder of acceptgiro's zijn mislukt of personen zijn kwijt of bedragen kloppen niet of mensen krijgen ten onrechte een herinneringsbrief thuis. Hier is dan geen discussie over. De applicatiebeheerders als 2^{e} lijns helpdesk bepalen of een issue een "problem" is, waarbij we altijd achterhalen of het een structureel kantje heeft. Gemeenten begrijpen tegenwoordig het systeem beter, dus als ze iets niet voor elkaar krijgen is de kans groter dat ergens een structurele issue in het systeem zit.

IT operations control: onderhoud plegen wij niet. Als het de hardware betreft doet Mendix dit. Wat data betreft wordt er netjes een backup gemaakt. Het enige dat wij met de data doen is af en toe een backup maken om te restoren op een testomgeving. In een Mendix gemodelleerde applicatie zit niet zo veel waarbij de applicatie output genereert waar je ook onderhoud op zou moeten plegen. Wel als bijvoorbeeld output wordt gegenereerd als functionaliteit. Dan is het interessant om regelmatig te kijken of de output gerenderd is. Voor een SaaS gebruiker vergt een Mendixapplicatie op dit gebied geen onderhoud; backups regelt Mendix en de gebruiker hoeft zich niet met de onderliggende Java-engine bezig te houden. Als je de server on premise zou hebben staan valt het onder het technisch beheer van de eigen organisatie en dan heb je hiervoor ook de expertise nodig. Wij hebben hier dus geen expertise voor nodig (maar hebben we wel in de vorm van Herbert).

ICT operations management: beschikbaarheid regelen we in de SLA met Mendix. Het zal ook voor een cloudoplossing gelden dat je een SLA hebt voor een bepaald percentage beschikbaarheid, waarna Mendix daar dus voor moet zorgen.
Als we meer ruimte nodig hebben krijgen we dat vanuit de hardware gemeld en vervolgens besluiten wij daarover, niet Mendix. We hebben wel eens meegemaakt dat de applicatie niet vooruit te branden was (vorig jaar vlak voor de kerkbalans), toen hebben we besloten even snel 30 GB RAM bij te prikken waarna hij het weer perfect deed. Vanuit technisch oogpunt gezien was er niets aan de hand, dus de monitoringtools zouden niets gemeld hebben (die monitoringdienst hadden we toen ook nog niet), want het systeem reageert. We hebben ook een eigen uitwijkomgeving die we kunnen promoveren tot productieomgeving en productie wordt dan uitwijk. Beide omgevingen worden beheerd door Mendix. Ik weet niet precies waar de productieomgeving en de uitwijkingomgeving staan, maar ze zijn in ieder geval gescheiden. Waarschijnlijk staan de servers fysiek bij XS4All in Amsterdam. Dat is een centrale locatie met een snelle internetverbinding vanuit de hele wereld en daarom kan het systeem ook goed op 10.000 gelijktijdige requests reageren.

Een belangrijk aspect is dat heel veel zaken in meer of mindere mate niet meer je aandacht nodig hebben omdat je ervan uitgaat (en dat kunnen ze aantonen) dat ze het zelf beheersen. Dus het technische beheer, het capaciteitsbeheer, monitoring kun je afnemen als dienst (in de cloud krijg je dat spontaan en gratis sowieso al in je omgeving erbij), toegang is in de cloud ook uniform geregeld (zelfs grafisch kun je precies instellen welke mensen wat wel of niet mogen doen).

Wat we een beetje missen is de mogelijkheid om een externe partij een soort “observer rol” of inzage te geven als we aan het ontwikkelen zijn, zodat ze kunnen meekijken, zien hoe we vorderen, zien wat we constateren en wat we doen (het hele ontwikkelproces). Zo’n inzagerol is er niet, gaat er geloof ik wel komen, hebben we ook al diverse keren om gevraagd. Binnen concerns is er bijvoorbeeld altijd een financiële baas of iemand die het naar buiten moet kunnen verwoorden die niets wil kunnen veranderen, maar wel alles wil kunnen zien en rechtsonder kunnen zien wat de status is. Je zou willen differentiëren tussen ontwikkelaar en meelezer. Intern zouden mensen dan kunnen zien wanneer hun issue aan de orde komt, wanneer je de gemeente moet inlichten, wanneer het probleem is opgelost zodat ze het kunnen gaan proberen. Deze cycle is problematisch als je het formeel wilt doen en zeker wilt weten dat niemand iets kan veranderen. Dit zou Mendix kunnen verbeteren.
Ten slotte wil ik nog zeggen dat het erg belangrijk is om niet te vallen voor het feit dat het zo snel gedemonstreerd kan worden en het ontwerpen dan ook maar daar wordt gedaan. Dit hebben we in het verleden gedaan en het zorgt bijna altijd voor uitloop in tijd en kosten. Wij hebben geleerd dat je het duidelijk moet definiëren, bij elkaar moet opsparen en daaruit vervolgend de keuze maken. Ook niet accepteren dat mensen onvolledig/onduidelijk zijn bij de aanmelding. Gebruiker duidelijk laten definiëren wat hij ervan verwacht, wat het nu doet en hoe hij het liever ziet of wat hij erbij zou willen laten maken en hoe het eruit moet zien; uniforme beschrijving. Dit strookt natuurlijk niet echt met de dynamiek van Mendix, maar je merkt dat je daar gewoon lelijk mee op je neus kunt gaan.
In het SaaS framework mis ik geen processen, er zijn zelfs een paar processen die nu zo zitten ingebakken dat je ze niet eens zou formaliseren in zo’n overzicht. Dit hebben we allemaal besproken.

Change Management
We hebben momenteel een eerste concept change management procesbeschrijving. Deze is opgesteld door de financiële man die bij het applicatiebeheeroverleg erbij zit. Hij wil voor alle applicatiebeheerders van alle afdelingen een standaardmethodiek waar we allemaal volgens gaan werken. Henry wil dit ook geharmoniseerd hebben. Dit proces is nog niet in effect. Het eerste voorstel dat er nu ligt is bijna letterlijk het traditionele ITIL procesmodel wat je hebt opgesteld en uitgeprint. Het proces is niet geoptimaliseerd voor Mendix, omdat we meerdere applicatiebeheerafdelingen hebben. Wij willen in zo’n proces wel softwarematig ondersteund worden; we willen daar een tool voor hebben.

Het proces: input komt (bij applicatiebeheer binnen) vanuit wetgeving, kerkorde, en economische veranderingen zoals SEPA (Europese harmonisatie van banken waardoor een heel nieuw betaalsysteem komt), vanuit interne afdelingen en het serviceapparaat (gebruikers, incident mgt., problem mgt.). Het serviceapparaat zijn 4 of 5 dames met een ‘teamhoofd’ die de telefoon bemannen. Ze beschikken over een kennisbank en nu na jaren ook over de nodige ervaring en een coördinator met meer rechten in het systeem. Het serviceapparaat is de eerste lijn en voert een eerste filtering uit. Als deze eerste lijn het niet kan oplossen dan komt het bij ons (applicatiebeheerders) terecht. De feedback is dan geregistreerd, wij beoordelen het en wijzen af/accepteren/markeren het als niet relevant of dubbel. We maken er dan een ‘issue’ van, waar meestal ook een stuk onderzoek aan vastzit en conclusie wat ermee moet gebeuren en de uitwerking ervan. We proberen zo zuiver mogelijk onderscheid te maken tussen een wens en signalering van een probleem dat het nodig maakt dat we daar snel een change op uitvoeren of het is een request voor change die duidelijk voor de toekomst pas speelt. Categorisatie en prioriteren is erg belangrijk, zeker als je veel werk te doen hebt. We vragen aan onze indiener (in principe het hoofd van de afdeling) om in zijn optiek een prioritering mee te geven. We houden hierin ook rekening met onze optiek en het kerkelijk bedrijf. Sommige dingen lijken geen groot issue maar vinden wij toch dat het niet moet voortbestaan. We hebben iets van 170 issues in onze werkvoorraad, daarvan kunnen we er ongeveer 20 wegwerken per 4 dagen. Het testen ervan duurt een dag. Er komen er per maand ook tientallen bij. De werkvoorraad lijkt nu over het algemeen een status quo te hebben, omdat we ook beter en gedifferentieerder wensen afvoeren. Uiteindelijk neemt de berg dus wel degelijk af.
De Mendix tool is heel sterk in het snel iets realiseren in de eerste fase, opleveren en het kunnen tonen hoe het gaat werken. Als je dat eenmaal voorbij bent en je hebt honderden zoniet duizenden vensters die allemaal werken, dan komt de fase dat er ook zo af en toe dingen bij optreden en daarin scheelt Mendix niet zoveel van andere tools; er moet gewoon onderzocht worden wat er aan de hand is (wat is het, ligt het aan de data, is het framework inefficiënt bezig, hebben we het datamodel verkeerd vormgegeven, hebben we de tabellen verkeerd structuur gegeven). Zodra je dus 20 of 30 issues in je bak hebt zitten moet je wel weer aan prioritering gaan doen wil je niet alleen de leuke dingen oppakken. Categoriseren hangt hiermee samen, omdat bijvoorbeeld issues uit de categorie die van de afdeling die voor de financiën zorgt een hoge prioriteit krijgen.

Ten behoeve van urgentie zijn er geen aparte procedures, het krijgt dan gewoon een hogere prioriteit. Het proces is dan dus hetzelfde, we gaan er alleen sneller mee aan de slag.

Plannen is niet nodig als aparte activiteit. Op basis van de SCRUM tool is, als je categoriseert/prioriteert het plannen daarmee ook gedaan. De prioritering dicteert de volgorde. Je zou issues kunnen differentiëren en toewijzen aan mensen die meer in die bepaalde hoek georiënteerd zijn en zich dus daar mee bezig kunnen houden, maar alles is dezelfde taal. Iedereen kan tegelijkertijd aan zijn zaken werken, zelfs als het correleert botst het in de meeste gevallen helemaal niet. Alleen voor low-level programmeren in Java heb je wel een specialist nodig.

Van het transformeren moet je je helemaal niets voorstellen. Dat is geen fase, het bevindt zich volstrekt buiten je invloedssfeer. Het gebeurt wel op de achtergrond, maar je hebt er niets mee te maken. Je modelleert en dat model test je, daar zit niets tussen. De server interpreteert het model en transformeert het naar Java-code, maar dat is totaal niet interessant. Wat wel interessant is, is het proces: lokale ontwikkeling (modelleren) -> deploy naar testomgeving -> testen -> corrigeren (= evt. terug naar ontwikkelen) -> acceptatie -> deploy op productie. ‘Ontwikkelen’ mag je ook ‘modelleren’ noemen. Het blijft dus eigenlijk build -> test -> implement. Model -> transform -> test -> deploy is een commercieel praatje.

Statusmonitoring door configuration management zoals wordt voorgeschreven door het standaard ITIL proces is niet relevant. Het framework verzorgt dit allemaal (database bijwerken, migreren, tabellen bijmaken, dingen hernoemen).

Je merkt dat de evaluatie-activiteit vooral belangrijk is voor externen die je erbij haalt. Die kennen het traditionele ITIL proces namelijk heel goed en willen dat heel graag. Deze mensen dreigen op dit punt te zeggen dat het niet geworden is wat ze verwacht hadden. In mijn optiek hebben we dan het voorgaande al niet goed gedaan. Bij ontwikkeling zoals de tool en ik dat voorstaan kan op dit punt geen evaluatie meer plaatsvinden, deze vindt namelijk plaats vóór en tijdens ontwikkeling. Bij het bouwen van nieuwe functionaliteit heeft de evaluatie van wat we gaan bouwen en hoe het eruit moet komen te zien al plaatsgevonden voor de request for quotation. Als het gebouwd is en het werkt en mensen hebben in de iteratie al kunnen zien hoe het gaat worden+hebben daar hun feedback op kunnen geven ga ik hier niet meer evalueren en de functionaliteit eventueel alsnog afvoeren. De fout die je niet moet maken is de uitwerking te veel aan de interpretatie van de ontwikkelaar overlaten door een te summiere omschrijving; je wil aan de voorkant al een zo duidelijk mogelijke beschrijving hebben, dus je moet niet letterlijk de melding van de eindgebruiker doorspelen. Een voorbeeld uit de praktijk van een eindgebruikerissue: “Het systeem moet intelligenter worden”. Als je dit Agile oppakt sta je voor een fiasco, want er gaan gegarandeerd vele iteraties overheen voordat deze gebruiker tevreden is. Dit moet je dus niet accepteren, het hoeft ook weer niet precies omschreven te worden, maar wel globaal en duidelijk. Dan kun je het Agile proces erop loslaten: issue lezen -> snel realiseren (binnen week) -> gebruiker mag erop schieten -> aanpassen naar aanleiding van evaluatie -> testen door gebruiker. “Je moet Agile niet gebruiken als excuus om maar ‘niets’ te definiëren, want dan kun je naderhand nooit meer terugvallen op wat nou echt de wens was”. De ontwikkelaar moet niet gedwongen worden in de rol van uitzoeken wat de klant wil, daar zijn businessmensen voor die er snel uit kunnen trekken wat de klant precies wil.
Verschillen change management tussen traditionele applicatie en Mendix applicatie: het erg veel registreren van versiebeheer en uitgebreide documentatie van wijzigingen (omdat het in de code zo lastig lezen is) is bij Mendix niet meer aan de orde. Bij Mendix kun je gewoon terugzien hoe het er in de vorige versie uitzag. Dankzij de Mendixtool hoef je niet eindeloos lang ingewikkelde plaatjes te genereren van hoe het er op databaseniveau en in de applicatie uit komt te zien, want het één resulteert bijna al in het ander. Als je het juiste ontwerp hebt gemaakt kun je heel snel wat dingen aanpassen, dat gaat vele malen sneller en is efficiënter. Ook het daadwerkelijk aanpassen gaat efficiënter en dan is het ook nodeloos om daar een volledig change management-achtige escapade op los te laten. Je kunt het de gebruiker laten zien en die kan besluiten dat het anders moet en dan kun je het ook direct anders maken. Het (ontwikkel)werk wordt met Mendix eigenlijk teruggebracht tot administratieve handelingen, als iedereen het er maar van te voren mee eens is dat “dit het echt gaat worden”. Server online brengen, zorgen dat de juiste software online komt, dat is allemaal niet meer aan de orde.

Er is een digitaal formulier voor registratie van issues, met bijv: “applicatieversie, browserversie, Windowsversie, wat had je verwacht, wat heeft het gedaan, welke knop heb je ingedrukt om het te reproduceren”. De browserversie is van zeker belang als je werkt met Mendix. XS2OS heeft een gestructureerd

format van welke informatie ze van ons willen hebben om een goed oordeel te kunnen geven over hoe lang de realisatie duurt en wat het gaat kosten (voor nieuwe functionaliteit of uitbreiding van functionaliteit).

Configuration Management
Op databaseniveau hoef ik als applicatiebeheerder weinig tot niets te doen. De tool voorziet in principe in alles wat op databaseniveau gebeurt. Mendix heeft richtlijnen voor naamgevingsconventies en als ontwikkelaars spreken we ook bepaalde conventies in het model af die we hanteren.

Mendix regelt configuratiebeheer voor de hardware. Het is niet mogelijk om losse componenten van de applicatie (service components) te registreren en gegevens+status hiervan bij te houden.

Wijzelf doen niet specifiek configuratiemanagement. Wij registreren in de applicatie in welke versie van het model een issue of functionaliteit gerealiseerd is. We hebben een productieomgeving en een testomgeving. De productieomgeving is altijd de laatste stand en de testomgeving loopt er één versie op achter. Daarnaast hebben we intern een proefomgeving die ook altijd de laatste versie heeft. We hebben momenteel ook een testversie voor onze testers waarvan we precies weten welke versie dat is. Meer hebben we aan versiebeheer niet te doen.
Als Mendix met een volledig nieuwe versie van de tool komt, heeft dat gevolgen voor bijv. je database, maar ook daar hoef ik niets aan te doen. Dat is een kwestie van “deployen en daar gaat ie”. Je zit er een halve dag naar te kijken en het ziet er spectaculair uit, maar behalve hopen en bidden dat er geen stoppende fouten in tevoorschijn komen hoef je niets te doen. Van tevoren heb je voorbereid en maatregelen genomen gebaseerd op het effect dat Mendix aankondigt. Bijvoorbeeld schijven erbij plaatsen als de database 4x groter wordt. Deze implicaties zijn de randvoorwaarden en verder hoeven wij er niet naar om te kijken.

Bij Mendix is configuration management voor ons niet van toepassing. Het ITIL procesmodel zoals je dat hebt uitgeprint wordt echt gemarginaliseerd. Ik kan er niets concreets bij bedenken. Mendix is op gebied van configuration management erg afgewerkt en houdt het voor ons erg buiten beeld.
De hardware is van ons, die hebben wij gekocht. Dit is gewoon standaard hardware, er is niets bijzonders aan. Mendix voert hier configuration management op uit. Mendix is zeker de partij die zegt: “als we dit framework succesvol willen laten draaien hebben we minimaal deze Linuxversie met eventueel die patch nodig en eventueel nog een hardening eroverheen”. Mendix houdt ook bij hoeveel servers er staan, met versiebeheer, en middels monitoring houden ze precies bij welke klanten op welk cluster zitten, uit welke servers dat cluster bestaat, hoe oud de servers zijn, hoe lang de schijven al meegaan. Als er een verandering doorgevoerd wordt de keuze maken: wordt het een NAS of een SAN of nieuwe hardware, dat soort dingen regelt Mendix allemaal. Wij spreken binnen SaaS helemaal niet meer over hardwarematig “wij willen dit of wij willen dat”. Bij SaaS is dat vertaald in slechts beschikbaarheid en performance (voor zoveel gebruikers). Als een Mendixapplicatie moet communiceren met een externe database van bijv. Exact of SAP is de harmonisatie daarvan de verantwoordelijkheid van Mendix.

Wij zijn de laatste jaren wijzer geworden wat betreft het Agile proces: laat het goed plaatsvinden. Zorg dat je goed in beeld krijgt wat de gebruiker wil, een ontwikkelaar mag best wat terugvragen, maar laat de klant goed zeggen wat hij wil. Als dat goed gedefinieerd is het je alleen maar lof van het iteratieproces. Want het kan heel snel werken, maar niet als je het je ontwikkelaar moet laten uitvinden. Mijn advies: laat het aan de voorkant zo duidelijk mogelijk wezen en vooral ook zo creatief mogelijk, bijvoorbeeld met een simpele tekening. De klant geeft dan een duidelijke richting wat hij wil en de ontwikkelaar is tevreden omdat hij voldoende vrij gelaten wordt, maar het wel duidelijk heeft. De ontwikkelaar weet dan ook dat hij straks waardering krijgt voor zijn werk, snel en efficiënt kan opleveren waardoor zijn baas tevreden is omdat hij binnen budget blijft en de klant is tevreden omdat hij het idee heeft dat hij werkelijk is gehoord.

LRP betrokken zijn de volgende mensen bezig: Support/helpomgeving: 6-8 man. Orderafdeling (zijn in principe interne afnemers): ong. 3 man. Applicatiebeheer: 2 man + 0,4 fte externe inhuur (Mendixspecialist). Dit allemaal onder leiding van het Hoofd ICT. De formele eigenaar is het hoofd afdeling institutionele ondersteuning, een dominee, die ook budgethouder is voor LRP.

APPENDIX H: CASE STUDY B

To perform this case study, interviews have taken place and documents that fall within the context of the research subject have been collected. The introduction text of the case study refers to external sources which have additionally been used to prepare the case study and verify some of the subjects that were mentioned in the interviews.

Interviews		
Code	**Position**	**Date**
B-I1	Manager Services Business Solutions	9-11-2012

Table 5: Interviews held in case study B

Documentation		
Code	**Title**	**Date**
B-D1	Nobel Incident management	1-1-2011
B-D2	Nobel Business Solutions Problem management	2010
B-D3	Nobel Business Solutions Change management	2011
B-D4	Nobel: Synergy Enterprise IT Service Management	2-11-2010

Table 6: Documentation used in case study B

The interview reports are not literal transcriptions of the interviews, but mentioned aspects are partly summarized and shuffled to present a more logical story. As much information as possible is included in the interview reports to not accidentally overlook important aspects that may have gotten lost when summarizing. The digital recordings of the interviews as well as the mentioned documentation are available for inspection with the author.

INTERVIEW REPORT B-I1

Interviewee position: Manager Services Business Solutions
Interview date: 9-11-2012
Interview duration: 185 minutes

Introduction
We hebben 5 beheerteams, waarvan 4 infra-gerelateerd (1 hier in Den Bosch en 3 in Naarden). Het team in Den Bosch heeft betrekking op alle business applications en de bijbehorende servicedesk heeft een eigen teamleider. Ik ben de manager van het business applications-team en daarnaast ben ik manager van de service level managers die wij zelf “Service Managers” noemen. Ik zit dus in twee hoedanigheden in dit bedrijf, enerzijds vanuit de software-kant en anderzijds overkoepelend. Overkoepelend zit ik bijvoorbeeld in het qualification board waarbinnen we aanbestedingen kwalificeren. Ik ben daar verantwoordelijk voor de input vanuit het beheer Nobel-breed. Ik ben dus niet continu met incident- en problem-processen bezig. Ik zit meer op strategisch/tactisch niveau.
Er werken bij Nobel ongeveer 330-370 mensen. Voor het gemak hou ik altijd 330 aan. Ik werk hier nu 3,5 jaar.

Ik wil graag benadrukken dat dit hele interviewverslag gebaseerd is op inrichting binnen Nobel en mijn persoonlijke mening. De inrichting van het beheer is afhankelijk van het platform dat je kiest, het advies dat de leverancier je geeft, de maturiteit van de organisatie en ten slotte van de implementatiewens van de klant.

Multi-client (Multi-applicatie)
De insteek is bij Nobel om vanuit de klantwens een applicatie te ontwikkelen of in te zetten. We bepalen dit dus niet vanuit een leverancier als bijvoorbeeld Mendix. Nadat we een applicatie hebben ontwikkeld zou deze in beheer genomen moeten worden. We hebben een generiek beheerproces ingericht. Dit werkt heel eenvoudig: een bepaalde klant heeft een bepaald incident en belt de service desk, de beheerafdeling. De Servicedesk doet de intake. Als het binnen de SLA valt wordt de service manager erbij geschakeld en als het niet binnen de SLA valt wordt het rechtstreeks doorgeschakeld naar de 2^{e} lijn, de Mendix consultants (= ontwikkelaars). Als de 2^{e} lijn er niet uitkomt hebben we natuurlijk altijd Mendix achter de hand als 3^{e} lijn, maar die zullen er ook niet veel mee kunnen omdat het de functionele inrichting is (een applicatie) die zij niet zelf

ontwikkeld hebben. Alle Mendix klanten die we nu hebben zitten allemaal in de SLA verweven. Dit betekent dat er dedicated een service manager op zit die de het beheerproces in de gaten houdt en de communicatie richting de klant voor zijn rekening neemt. Als een bug met betrekking tot het Mendixplatform bij de service desk binnenkomt zetten we deze rechtstreeks door naar Mendix en als het een infrastructuur-probleem is en het draait in onze eigen cloud (IaaS) gaat het door naar ons cloud team. Alle Mendixapplicaties die we nu hebben draaien in onze eigen cloud.
De reden voor deze procesgang is dat alles (resp. applicaties, platforms) ontzettend complex is en wij al die kennis nooit op dat niveau allemaal hier (binnen het beheerteam) kunnen opbouwen. Vragen/bugs over bijvoorbeeld het Mendixplatform gaan door naar Mendix. Dat is hun verantwoordelijkheid en staat ook in de onderhoudscontracten. Als het de functionele inrichting (applicatieniveau) betreft moeten we het zelf oplossen.
Eigenlijk zou je willen dat de 1e en 2e lijn geïntegreerd zijn in één afdeling, want eigenlijk hoef je voor Mendix alleen de manier van “procesdenken” te kennen. Ik ben echter van mening dat dat een utopie is, het is namelijk niet alleen het proces, maar ook de techniek. Naast het modelleren in Mendix zitten er bijvoorbeeld heel vaak ook scripts (Java code) bij. Het komt namelijk voor dat bepaalde dingen niet mogelijk zijn binnen het Mendix platform en dit moet dan alsnog geprogrammeerd (gecodeerd) worden. Ook is het onmogelijk om iedere specifieke klantsituatie binnen dit beheerteam te kennen. Het is dus naïef om te denken dat alles binnen het beheerteam (1e lijn) kan worden opgelost, maar door daar de Mendix-ervaring op te bouwen kun je er wel zoveel mogelijk al oppikken. Bovendien kan de 1e lijn direct betere vragen stellen aan de klant, waardoor wij het sneller het antwoord kunnen vinden.

De applicaties die wij momenteel in beheer hebben zijn door onszelf gebouwd. Dit zijn er een stuk of 15 gebaseerd op Mendix, op een stuk of 5/6 grote omgevingen. Hierbij zit ook een EDI applicatie voor berichtenverkeer tussen een leverancier en een klant. Dit hebben wij op het Mendixplatform gebouwd. We zijn momenteel ook bezig met de ontwikkeling van twee nieuwe omgevingen.

Nobel is naast Mendixpartner bijvoorbeeld ook partner van Exact (we doen bijv. Synergy en Globe implementaties). Er is dus ervaring met beheer van verschillende applicaties, zowel on-premise als SaaS. Bij de ontwikkeling van applicaties wordt ook afgewogen wat het best zou passen. We hebben in plaats van Synergy wel eens gekozen voor Mendix, omdat Synergy bijv. veel meer functionaliteit biedt dan vereist was. Ons advies aan de klant is volgens mij in twee gevallen omgebogen van Synergy naar Mendix. Voor al onze beheeractiviteiten hanteren wij in principe gewoon de conventionele ITIL processen. Die hebben we beschreven. Het is bij ons tegenwoordig heringericht naar ITIL versie 3. Sinds 2010 zijn wij partner van Mendix, we zijn daar dus vrij vroeg ingestapt. Zowel ontwikkeling als beheer zijn we mee begonnen in 2010. De eerste applicaties die we ontwikkelden waren kleine cases. Tegenwoordig heeft Mendix zijn licentiemodel aangepast en focussen ze zich niet meer op MKB, maar op MKB+ en corporate. Met name de eerste applicaties die wij ontwikkelden waren klein MKB. De tijd die het kost om een Mendixapplicatie te ontwikkelen hangt volledig af van de grootte en heeft alles te maken met de functionele vraag van de klant. Het aantal gebruikers maakt hiervoor niet uit, dat zit gewoon in de licentie. De Mendixapplicaties die wij momenteel in beheer hebben zijn voornamelijk grote applicaties die bedrijfsprocessen ondersteunen en auditen. De applicaties die wij beheren variëren van 5 tot 100 gebruikers. Door het nieuwe licentiemodel van Mendix zal het meer naar het hoger segment toegaan en dan komen situaties met maar 5 gebruikers steeds minder voor.
De grootste waarde van Mendix vind ik dat de klant wordt betrokken bij het ontwikkelproces. De klant ziet na 2 of 3 weken gewoon al resultaat. Vaak wordt eerst een deel van de applicatie ontwikkeld voor een deel van de klantorganisatie en wordt het daarna uitgebreid met een volgend deel. Iteratief werken is in 1995 met DSDM natuurlijk al bedacht. Agile is daar gewoon een populaire term voor en heeft het iets gefinetuned, maar eigenlijk is het niets nieuws. Een workflowsysteem wordt vaak op deze manier ontwikkeld. Het wordt in stukken gehakt om het overzichtelijker te maken en in wezen werk je dan al iteratief.
De gebruikers van onze applicaties zijn de end-users bij de betreffende klanten.

Management
Er is een incident management proces ingericht zoals eerder besproken. Voor het Mendixbeheer is dit het belangrijkste proces. Er is ook een change management proces dat door/via de Mendix consultants zelf wordt gedaan. De applicaties worden vaak gefaseerd opgeleverd en daardoor zijn er continu consultants mee bezig die ook de gewenste wijzigingen meenemen. Dit gaat dan dus niet via een conventioneel change proces, er wordt vaak wel analyse gedaan maar er wordt geen RfC geschreven. Dit proces zit meer in de problem solving. Voor conventioneel beheer hebben we standaardprocessen (bijv. change, problem en incident, configuration

mgt). Qua configuration management registreren wij van onze klanten welk versienummer van de omgeving/applicatie ze gebruiken. Dit wordt niet bijgewerkt door onze consultants, dus we werken dat bij vanuit hun verslagen en vaak moeten we het toch nog aan de klant vragen. Voor Mendixbeheer doen we geen configuration management en houden we dus ook versienummers niet specifiek bij. Bij Mendix ontwikkeling maken wij gebruik van een OTAP (Ontwikkeling Test Acceptatie en Productie)-straat waarbinnen de configuratie vastligt. Zij kunnen deployen vanuit hun eigen omgeving.
Wij hebben de processen via ITIL ingericht. Onze mening is dat dit als het ware een samenvoeging is van veel case studies waar je gebruik van wilt maken. Ook ons hele ISM (Integrated Service Management) systeem, waarin wij onze callregistratie doen is volgens ITIL ingericht met de bijbehorende procesafhandeling. Procesafhandeling is leuk en aardig, maar je kunt “in the end” kwaliteit leveren door: een band te hebben met je klant en te weten wat die klant wil en heeft staan en daarop acteren. Under-promise, over-deliver: realistische afspraken maken en deze afspraken nog beter oplossen. Onze inrichting van service management is ook zo ingericht. Echte (grote) klanten hebben SLA’s; het wordt via het beheerteam afgehandeld en service managers springen bij. Mochten we dan bijvoorbeeld een consultant nodig hebben die op het project zit, dan gaat de service manager die persoon van het project afhalen.
De processen zijn richtlijnen (hoe je werkt), maar in the end zit de kennis natuurlijk in de persoon. Een Mendix consultant weet bijvoorbeeld precies wat er draait. Het “hoe ga je met je klant om” is belangrijker dan processen. De klant eerlijk vertellen dat hij niet alles kan krijgen als hij er niet voor wil betalen en issues verhelpen van klanten die geen SLA hebben zijn daar belangrijke voorbeelden van.
We hebben sinds vorige week een vastgesteld Change Management proces in gebruik genomen. Service managers kunnen nu ook hier bij de servicedesk RfC’s aanmelden en worden RfC’s door de beheerconsultants (service desk) geschreven. Met problem management zijn we ook bezig. Een aantal klanten nemen dat als dienst bij ons af en we hebben er nu ook een problem manager op zitten. Vanuit de problem manager worden recurring incidenten geanalyseerd en opgelost. Deze informatie gebruiken we dan voor die klant of andere klanten. Een incident is een problem als het vaker voorkomt. De oplossing voor een problem (of known issue) staat in onze kennisdatabase, waar de service desk direct bij kan en dus de oplossing kan noemen na goede analyse. De kracht van incident management is zo snel mogelijk het incident oplossen en bij problem management kan het zijn dat het probleem morgen weer terugkomt en je er langer over doet om het op te lossen. Bijvoorbeeld omdat het op verschillende niveaus speelt. Als het bij meerdere klanten optreedt worden deze incidenten ook gebundeld en netjes in een problem samengevat. Vaak heb je tussendoor een workaround. Door middel van problem management kun je erachter komen waar het probleem zit, als meerdere klanten het ervaren dus bijvoorbeeld in het platform i.p.v. in de applicatie. De consultants hebben dit vaak al snel door omdat zij er continu mee bezig zijn.
We gebruiken nu de ITIL processen en rapporteren dus KPI’s op basis van incident management, maar eigenlijk vindt de klant dat helemaal niet interessant. De klant wil weten of zijn proces loopt en als het niet loopt waarom het niet heeft gelopen. Daarom gaan we nu ook met BiSL aan de slag; in plaats van vanuit specifiek service management meer naar het bedrijfskundig niveau. We gaan dus meer naar business KPI’s toe in plaats van operationele KPI’s.
Iedereen binnen het bedrijf kan te maken hebben met de beheerprocessen. Dit is afhankelijk van het probleem. Normaalgesproken zijn de actoren: de teamleider van de servicedesk, de servicedeskmedewerker. Bij een groter probleem komt de escalatie bij mij terecht, nog groter dan gaat het naar mijn manager toe en nog groter zou het wel eens bedrijfsbreed opgepakt kunnen worden. In het geval van Mendix heb je ook te maken met de Mendix consultant en diens manager. Dit zijn de eerste en tweede lijn. Belangrijk is dat hier service management (de service manager) overheen zit. Incident management doen we in principe vanuit de eerste lijn. Dit team van 5 personen is eindverantwoordelijk voor de totale workflow aan incidenten die openstaan en is verantwoordelijk voor de dispatch naar het juiste team. Dit laatste is best ingewikkeld, want we hebben 14 productlijnen (applicaties), van logistieke applicaties tot financiële applicaties en business intelligence applicaties. Dit worden er meer, we gaan naar de 20. Bij het dispatchen moet bijvoorbeeld rekening gehouden worden met of we de hosting zelf doen (dan moet het naar een beheerteam), of het naar een leverancier toe moet (bijv. hosted desktop). Inclusief leveranciers is incident management bezig met misschien wel 40-50 lijnen binnen 150 incidenten die momenteel openstaan. De incident manager moet dus echt weten hoe het werkt, wat de afspraken zijn, wanneer hij de klant moet informeren, wanneer hij de interne organisatie moet aanschrijven en wanneer het moet escaleren. Dit is dus een hele belangrijke actor.
ITIL is niet gebaseerd op agile ontwikkeling. In ITIL v3 zijn ze vanuit support- en deliveryprocessen naar een lifecycle gegaan. Dit vind ik sowieso veel beter. We hebben het service management proces (= redeneren vanuit bedrijfsoogpunt), incident management, problem management en change management in place. We zijn bezig met een proces waarmee we nog meer ITIL processen gaan beschrijven indien die er op dit moment

zijn. Bijvoorbeeld availability, event en capacity management doen we maar hebben we nog niet op papier staan. We weten wel hoe we het doen, maar dit is nog niet beschreven, dus niet voor alle klanten hetzelfde. Het varieert hierdoor per beheerteam. We willen het standaardiseren en uniformeren om te kunnen weten wat we überhaupt doen en eigen interpretaties tegen te gaan, zodat alle klanten hetzelfde geholpen worden. Er is een link tussen Capacity en Event management. Als je bijvoorbeeld een threshold op 80% van de storagecapaciteit zet wordt op het moment dat dat bereikt is een event gegenereerd waarna vanuit het capacity proces storage wordt toegevoegd aan die machine. Hierbij hoort ook het accountproces: door middel van terugkoppeling naar de klant aangeven dat in plaats van ruimte bijplaatsen er misschien ook opgeruimd kan worden. Storage bijplaatsen kost geld dus daar zit het offerteproces weer in verweven. Het beheerproces heeft hier met het salesproces te maken. Wij hebben een eigen cloud (IaaS) in ons eigen datacenter. Deze gebruiken we ook om Mendix te hosten. Voor bijna al onze Mendixklanten hosten wij zelf, van één klant weet ik het niet zeker maar die zit volgens mij ook niet in de Mendixcloud, dus de hosting wordt daarvoor door een andere provider gedaan. Voor conventionele applicaties hebben wij vaak te maken met een alternatieve ICT partij, in het geval van Mendix kan dat ook maar zijn er meestal geen externe partijen. Nog niet. Dat moet ook nog groeien, die markt is natuurlijk ook nog kleiner dan bijvoorbeeld de Exact-markt in Nederland. Vaak is het de keuze van de klant aan welke partij ze wat uitbesteden. Bij de ontwikkeling op Mendix offreren wij natuurlijk ook onze eigen cloud mee, maar als het goedkoper kan bij een ander kan de klant daar ook heengaan. Je ziet bij kleine klanten ook dat ze graag iemand hebben die bij ze in de buurt zit, die is voor het gevoel sneller bij je. Grote klanten hebben daar geen last van en regelen hun hosting via een aanbestedingstraject.
Wij hebben SLA's met onze klanten over de functionele inrichting, hierin staat wat voor beheer wij uitvoeren en we hebben een underpinning SLA met Mendix. Dit laatste heeft natuurlijk geen zin op het moment dat een incident niet over het platform gaat, want dan is het geen verantwoordelijkheid van Mendix en kunnen ze wel ondersteunen maar zal dat nooit op SLA-basis zijn. Bugs, updates, etc. voor het platform zijn geregeld in de SLA met Mendix. De bouw (=functionele inrichting) ligt bij ons en daarover maak ik SLA afspraken met de klant. De klant heeft dus geen SLA met Mendix, wij zijn de Mendix partner en hebben die SLA. Mendix heeft een standaard SLA waarin zij zich aan bepaalde, strakke oplostijden conformeren. Als een klant dus in de Mendixcloud zou zitten hebben ze hier automatisch mee te maken.
Randvoorwaarden voor het beheer zijn: goede afspraken met de klant (verwachtingsmanagement), dit zit in de SLA. Een eis is eigenlijk ook een OTAP-omgeving waarin je deployment kunt regelen om te ontwikkelen, testen en accepteren. Dit zit ook in de Mendix-omgeving. Dit is ook belangrijk voor de klant als wij de ICT niet doen. Klanten kunnen specifieke randvoorwaarden hebben, hiervoor heeft de SLA specifieke paragrafen.
Bijvoorbeeld als je te maken hebt met een derde applicatie waarin een koppeling maar Mendix wordt gemaakt. Dan kunnen wij bijvoorbeeld het supplier management (third party management) doen. Als de interface tussen beide applicaties dan niet werkt mogen ze het bij ons aanmelden en zoeken wij uit of het aan ons ligt of aan de derde partij, waarbij we in het laatste geval pro-actief die derde partij inschakelen. Dit hebben we ingericht bij een Mendixklant met een koppeling naar Synergy dat wordt beheerd door een andere partner.
De change, problem en incident management processen zijn geautomatiseerd op basis van een workflow. Nadat het is ingetypt, wordt het gedispatht naar een bepaalde oplosgroep. Hiervoor hebben we het service management systeem. Monitoring (bijv. middels een agent applicatie) van de hardware en het netwerk zijn ook geautomatiseerd. Het netwerk wordt door een connectivity team in de gaten gehouden en events door alle beheerteams voor hun klanten. Deze automatiseringsoplossingen werken allemaal niet perfect samen, maar ze zijn wel geautomatiseerd. Eigenlijk heeft dit niet veel met beheer te maken, het is meer fault management waarmee je kunt zien dat er iets niet goed gaat en daarop ga je acteren met bijv. beheer of bezoek van een accountmanager. Dat ligt eraan wat de afspraken zijn, welke SLA's eraan ten grondslag liggen. Als er bijvoorbeeld alleen maar een helpdesk/incident contract is (relatief goedkoop) is de oplostijd best effort.
Wat betreft reporting worden er periodiek SLR's (Service Level Reports) gemaakt voor de SM (Service Management) klanten. Dit is een belangrijk rapport. Andere klanten krijgen geen reports. Binnen applicaties of op het Mendixplatform worden geen reports getoond. Ik weet wel dat Mendix een bepaalde monitorfunctie heeft. We hebben daarnaast nog meer reporting voor intern gebruik. Voor wat betreft Mendix kunnen we terugzien welke incidenten gemeld zijn en hoe ze opgelost zijn, daarnaast worden ook changes gemeld in de SLR. Je zou bij Mendix zelf kunnen vragen hoe hun reporting werkt. Mendix doet een heel klein stukje van het beheer. Ze doen beheer op hun eigen omgeving, maar geen support op de inrichting.

SaaS management framework
ITIL is ons framework. Daar refereren we aan, dus hebben we niet zelf een framework vastgelegd.

In financial management is het verschil dat de klant nu (SaaS) een dienst afneemt die in een flexibele licentie is gegoten en maandelijks opzegbaar is in plaats van afspreken wanneer je de klant factureert (vooraf of voor een bepaalde periode), dus daar zit een iets ander contract aan vast. Je hebt met een SPLA (Services Provider License Agreement) te maken, dat is meer flexibel. Je ziet ook dat scaling in financial management terugkomt. We gebruiken een omgeving die heet “Cortex”, een auto-provisioning omgeving. Daarmee kan de klant zelf zijn users toevoegen. Vanuit die omgeving factureren wij ook, afhankelijk van het aantal gebruikers en welke applicaties die gebruikers gebruiken. Dat is heel flexibel. Voor Mendix is dit niet van toepassing, Mendix gebruikt gewoon een userlicentie. Vroeger was dit als volgt: small business 10 licenties, medium 50 en enterprise op een aparte offerte, maar volgens mij hebben ze dit nu aangepast. Vanuit de cloudomgeving kun je ook in Mendix heel makkelijk scalen. Ik denk dat een beschrijving van het proces (financial management) heel interessant is. Dit denk ik van alle processen. Belangrijk is ook de samenhang tussen processen. Ze hebben altijd met iets anders te maken. Contract management, financial management en subcontractor management bijvoorbeeld. IT service continuity management heeft ook met financial management te maken: praten we bijvoorbeeld over disaster recovery of is er een backup. Binnen ITIL zijn de processen gedefinieerd, maar is ook de samenhang weergegeven en dat is een belangrijk aspect van het framework.
Hosting is een gestapelde dienst en een samenwerking binnen Nobel. Bijvoorbeeld een managementoplossing met eigen cloudhosting heeft met 3 beheerteams te maken: het cloudteam, het infrabeheerteam en het business applications team van hier. Zeker met betrekking tot Mendix is de afspraak dat alle incidenten hier (business applications team resp.) terecht komen; altijd aan de bovenkant, want de klant kan niet zien wat er fout is, alleen dat de service niet werkt. Hier wordt dan met een primaire analyse het probleem gepinpoint. Als het probleem niet in de applicatie, maar in een onderlaag zit wordt het naar het technisch beheerteam (=infrateam; business systems) gedispatcht en die stemmen dan ook af met het cloudteam.
Demand management ligt als ik het goed heb bij het technisch beheerteam, maar dat weet ik niet 100% zeker. Het kan ook bij het cloudteam liggen, maar dat maakt ook niet zo veel uit denk ik. Het proces demand management is op strategisch niveau volgens mij meer het in gesprek gaan met de klant over hoe hij met zijn data en gegevens omgaat. Dat leg je vast in contracten en je definieert hoeveel capaciteit de klant op dat moment krijgt en dat je de threshold op 80% zet en daarna samen met de klant verder kijkt hoe de ontwikkeling van zijn data loopt, zodat er nooit een probleem is met de continuïteit van de dienstverlening. Het proces demand management gaat om het voorkomen dat de storage volloopt en daardoor niets het meer doet. De strategische processen zie ik allemaal voornamelijk als samen afstemmen met de klant en de output daarvan.
Service catalogue management: wij hanteren een PDC (producten- en dienstencatalogus) en daarin staan onze diensten beschreven. Vastgelegd is welke diensten, wat ze kunnen verwachten en wat daarbij komt kijken. De term “service” is in deze context verwarrend, omdat een applicatie normaalgesproken een product wordt genoemd en bijvoorbeeld het management een dienst. Dit onderscheid verlies je als een applicatie ook dienst wordt genoemd.
Mendix is niet meer dan een ontwikkelplatform. Dat doet eigenlijk niets, je kunt er dingen op bouwen. Je bouwt dus een applicatie op basis van Mendix. Die applicatie wordt gedefinieerd in de PDC. Mendix biedt zelf ook geen applicaties aan en heeft dus ook geen service catalog te vullen. Wij hebben ook geen service catalog met specifieke Mendixapplicaties, want eigenlijk alleen EDI bieden we als standaardproduct aan. Dit wordt natuurlijk ook niet als “Mendix” verkocht, maar als de applicatie “EDI”. Mendix is wel faciliterend om het product aan te kunnen bieden. Sommige bouwstenen (services) zou je voor gebruik in grotere applicaties kunnen aanbieden, maar het gros van de Mendixapplicaties is maatwerkoplossing (een specifieke invulling voor een specifieke klant), die je daarom niet zomaar aan andere klanten kunt aanbieden. De klant heeft altijd een functionele vraag waaraan middels een maatwerkoplossing door een leverancier word voldaan, dus naar mijn mening werkt het andersom. Vanuit de klantvraag ga je de requirements analyseren en daarvoor definiëren wij een oplossing. Dat hoeft niet vooraf te betekenen dat het Mendix wordt. Een klant zal ook niet naar ons toekomen met de vraag om de catalogus te bekijken, maar met een specifieke klantvraag. Ik zou daarom eerder iets als “requirements management” verwachten op dit niveau. Je gaat met Mendix namelijk heel specifiek kijken naar de klantvraag. Ik denk dat we in de toekomst ook naar “templates” toe zullen gaan in Mendix, bijvoorbeeld een logistieke template en financiële template om standaard aan te leveren voor een bepaalde bedrijfsgroep. De kans is dat er een standaard template ontstaat als je een bepaald proces vaker bouwt in Mendix. Requirements management is meer een benadering vanaf de voorkant: wat nieuw te ontwikkelen. Service catalog is meer redeneren vanuit wat er al is.
Contract management: er is een contract met de klant.
Subcontractor management: bijvoorbeeld een contract met Mendix op het moment dat wij de applicaties zouden hosten op de Mendixcloud. Als je van het Mendixplatform gebruikmaakt heb

onderhouds/leveringsvoorwaarden. Dit is een standaardcontract tussen de leverancier en de partner. Subcontractor management is bijvoorbeeld aan de orde als een klant een bepaalde vraag heeft die we met Mendix voor een groot gedeelte kunnen invullen en het resterende deel met een ander pakket. Bijvoorbeeld een pakket waarin je producteigenschappen variabel kunt bijhouden, omdat deze waarden in Mendix hard-coded zijn. Je zou dan voor iedere (prijs)wijziging een nieuwe release van je Mendixapplicatie moeten definiëren. Door gebruik van een alternatief pakket is dit niet meer nodig. Als dit pakket door een andere leverancier wordt geleverd heb je een subcontractor.
Service level management is een heel belangrijk proces binnen de service design fase en deze mis ik in het framework. Ik ben het er niet mee eens als dit is vervangen door contract management omdat ik contract management meer het bewaken van het contract vind. Service level management doet echter meer: hij is mede verantwoordelijk voor additionele omzet, hij komt bij de klant over de vloer, hij ziet dingen die er niet goed gaan, hij hoort dingen, hij zit in service level meetings, hij leidt escalaties, wij plannen bovendien consultancy vanuit de service manager. De service manager kan gewoon veel meer waarde toevoegen dan alleen het stukje contract naleven. Ik wil de Service (level) managers zoveel mogelijk bij de klant zitten en bij de klant werken: zichtbaarheid bij klanten is belangrijk. Bij @Home had ik een service manager en een contract manager. De contract manager was voornamelijk intern gericht en de service manager was naar buiten, naar leveranciers, gericht. Het is een twee-eenheid in wezen, maar de één is gericht naar de klant en de ander naar de leverancier toe. Vooral de gevestigde term “service level management” vind ik belangrijk. Neem dat in overweging. Zowel subcontractor als contract management kunnen in wezen supplier management zijn, dus die zou ik eerder samenvoegen.
Wat betreft change evaluation heb je in een traditioneel ITIL change management proces een CAB (Change Advisory Board) gedefinieerd waar mensen vanuit diverse disciplines kijken of een change goedgekeurd moet worden of niet. Het traditionele proces release management definieert daarnaast in welke release welke changes worden gedaan. Een change evaluation proces voegt in dat geval niet zoveel toe. De CAB moet de gapanalyse tussen de IST en SOLL situatie bespreken, resulterend in een RfC of RfC’s. de CAB bepaalt ook of een RfC noodzakelijk is. Ik kan me wel voorstellen dat je in de gaten wilt houden wat de conclusie daarvan is, maar ik zie dat alleen niet in de praktijk gebeuren. Het is al moeilijk om een CAB in te richten, waardoor changes wel eens on-the-fly worden uitgevoerd. Theoretisch zie ik hierdoor wel toegevoegde waarde, maar in de praktijk weinig. Ik vind evaluaties eigenlijk altijd nodig, maar als je in de praktijk kijkt wordt het eigenlijk altijd overgeslagen. Iedereen heeft het over “lessons learnt”, maar als er al een lessons learnt-meeting wordt belegd wordt er daarna helemaal niets met de output daarvan gedaan.
Software management hangt sterk samen met configuration management. De cloud is meer een fundering van servers, hardware en operating systems. Daarbovenop komen de applicaties. Op applicatieniveau kun je verschillende omgevingen hebben en moet je toch aan distributie doen (van bijv. updates). Bijvoorbeeld voor alle applicaties die een klant op zijn cloudomgeving heeft staan. Waar je nu bijvoorbeeld 3 systeembeheerders nodig hebt om de hele omgeving in de lucht te houden heb je in het geval van SaaS één mannetje die voor een aantal klanten het systeembeheer doet vanuit een cloud. Het wordt efficiënter en het principe verandert, maar het doel blijft gelijk: dat er iets is waarop de applicatie kan draaien. Op dat moment is er nog steeds distributie nodig. Dit geldt voor een private cloud. In het geval van een shared cloud doe je ook distributie. Hierbij heb je ermee te maken dat je op het moment dat je de distributie doet, het voor alle klanten gelijk is. Het heeft zijn weerslag op alle klanten. Je zult als je iets doorvoert dus moeten checken of daar geen omgevingen mee omvallen. Distribution management kun je denk ik niet zien als onderdeel van change management. Het is echt de uitvoering en change management is het analyseren van de consequenties. De cloud impliceert al dat het meerdere omgevingen zijn; het is een wolk en je weet niet waar het draait. Dit is ook van belang als je naar wetgeving kijkt. Sommige klanten willen gegarandeerd hebben dat het in Nederland draait. Technologisch maakt het niet uit waar het staat. Klanten willen daarnaast dat er op een bepaalde manier met de data-integriteit omgegaan wordt: zaken als backups, disaster recovery. Dat hoort volgens mij ook op dit niveau thuis. Kan ook op operationeel niveau of strategisch niveau. Ik denk dat dit meer een strategisch proces is: de strategische afweging kan zijn dat de data op meerdere plekken staat, het meer geld kost, hoe de synchronisatie geregeld wordt, hoe snel de applicatie weer in de lucht is als een vliegtuig op datacenter 1 valt (recovery times).
Voor wat knowledge management betreft krijgen we met Exact bijvoorbeeld alle incidenten door en de oplossingen erbij, dus dat is eigenlijk ons knowledge systeem. Sowieso is de kennis aanwezig bij mensen.
Access management is het toegang geven van gebruikers tot een bepaalde omgeving, hiervoor zijn procesoplossingen. Ook binnen Mendix. Toegang beheren is iets wat klanten heel vaak zelf doen: toegang geven, rechten en rollen toekennen. Bijvoorbeeld bij een Synergy implementatie waar een applicatiebeheerder op zit dan doet de applicatiebeheerder (die bij de klant werkzaam is) dat. Wij bieden wel de dienst “nieuwe

user aanmaken", maar dat doen we bijna nooit. De applicatiebeheerder geven we verregaande rechten. Identity management hebben we het hier voornamelijk over, dat zouden we ook hier bij de service desk neer kunnen leggen als de klant dat wil. Als ik de klant was zou ik ook zelf de regulatie in de hand willen houden, wie toegang heeft. License management hoort hier ook bij. Het heeft er een relatie mee omdat je soms eerst de licentie moet uitbreiden voordat nieuwe gebruikers toegevoegd kunnen worden.
Event management kan ik je vertellen hoe dat gaat met betrekking tot infra. Event management op basis van applicatiemonitoring is echter iets wat niet heel vaak voorkomt. We zijn er wel mee bezig, maar monitoring op procesniveau is er niet. Ik weet dat er een bepaalde monitoringfunctie is binnen Mendix, maar ik weet niet hoe die werkt.
Service lifecycle management hoort volgens mij eerder thuis in de continuous service improvement fase dan in de service operation fase. Deze fase (continuous service improvement resp.) is natuurlijk de pijl eromheen, het gaat over alles heen. Je bent namelijk continu bezig met je dienstverlening en de aangeboden services bezig. Zeker als je een service manager hebt, wiens taak dit is, kan ik me iets voorstellen bij dit proces. Het PDC gaat bij ons via "business development". Daar ontwikkel je nieuwe diensten die eraan toegevoegd worden. Daarna kan het verkocht worden door sales en moet het ook beheerd worden door beheer en moet de project manager het kunnen implementeren. Volgordelijk is het sales -> delivery -> services -> beheer. Als dit stukje is opgeleverd en het live staat komen er op een gegeven moment toch weer vragen/requirements/aspecten om de hoek kijken om die dienst uit te breiden. Als de klant meer over de basis van incidenten wil weten, dan gaan we bijvoorbeeld met problem management aan de slag. Het proces service lifecycle management zal ook een directe link hebben met change management denk ik, als je verbeteringen redeneert. Je bent natuurlijk wel bezig met toevoegingen (bugfixes of nieuwe functionaliteit) waarna er een nieuwe release komt. Dit is meer van toepassing op de omringende beheerdiensten en het heeft een directe link naar strategie. Ik vind een dergelijk proces heel belangrijk, omdat je continu met de klant bezig bent. De klant zit te wachten op een leverancier die met hem meedenkt. Wij moeten ook mee blijven kijken naar nieuwe ontwikkelingen in de markt, functionele vraag van de klant en daar zo actief mogelijk op acteren, zelfs zonder dat de klant echt een vraag stelt. Actief meedenken met de klant vind ik "in the end" een belangrijk aspect in alle beheerprocessen, met name van de tactisch/strategische processen. Daar onderscheid je je juist door pro-actief gedrag.
Use support klinkt misschien vriendelijker naar de gebruiker toe, maar incident management is wel dé verankerde term in de IT. Use support zegt mij namelijk niets, ook niet nadat je mij het verschil hebt uitgelegd. ASL heeft qua borging nog lang niet de breedte die ITIL heeft. Als ik iets nieuws geborgd zou willen krijgen (nieuw framework resp.) zou ik iets herkenbaars gebruiken, waar mensen snel mee kunnen identificeren. Daarom zou ik kiezen voor de gebruikelijke term "incident management". Als je door pro-actief frontmanagement echt probeert incidenten te voorkomen voegt het wel iets toe, maar ik vraag me af of je het dan use support moet noemen. Ik zou het proces incident management missen in het framework.
Change/incident/problem is natuurlijk altijd al een drieluik geweest bij elkaar. Incident management maakt in principe ook al onderscheid tussen reactief en proactief, je wilt eigenlijk dat de klant het incident niet eens merkt. Je wilt het zelf signaleren door bijvoorbeeld event management en het eigenlijk oplossen voordat de klant je belt of tijdens dat de klant je belt.
Als je kijkt naar ITIL v2 of v3 beginnen alle processen vanuit strategie. Als je kijkt wat alle bedrijven in de praktijk doen is dat beginnen met incident management (operatie). Waarom: omdat het makkelijker is dan strategie bepalen. Na incident komen dan problem en change erbij, terwijl er eigenlijk vanuit een strategie gewerkt zou moeten worden.
Request fulfillment ken ik. RfI's (Request for information) kun je bijvoorbeeld in een request fulfillment kwijt. Wij hebben bijvoorbeeld klanten die vragen wat de consequenties zullen zijn als ze van Windows 7 overstappen naar Windows 8. Dat is in wezen een RfI. Daar moeten wij tijd in stoppen en geld voor vragen. Je levert een analyse (of rapport) op, waarin de klant geïnformeerd wordt over de vraag die hij had. Dit kan bij ons via de service desk, via de service manager, via de beheerder, via engineers, via consultants binnenkomen. Vaak zal het wel binnenkomen via de servicedesk. De klant stelt ze dan een vraag en die wordt gelogd als "information request". Het is belangrijk om dit in een apart request fulfillment proces onder te brengen en niet door incident management af te laten handelen. Als dit namelijk in je incident management proces zou zitten zit er bijvoorbeeld de prioriteit "information request" aan vast waar je meer dan 40 dagen voor hebt. Het wordt dan dus gezien als iets wat helemaal niet belangrijk is, terwijl het eigenlijk wel heel belangrijk kan zijn.
Problem management is juist niet op snelheid. Het is in wezen ook geen kwantitatieve KPI. Het heeft dus ook niet met de SLA te maken, dat doet incident management. Problem management heeft puur te maken met een recurring iets dat je gaat analyseren waar het vandaan komt. Vaak komt er een tijdelijke workaround en resulteert een problem in een change, een wijziging. Daarna is het incident weg, problem weg en heb je een change. Ik vind het moeilijk te zeggen of dit voor SaaS-applicaties ook zo zou moeten werken. Je kunt met

Mendix namelijk na één sprint al een stuk van de applicatie opleveren en in beheer nemen terwijl de rest van de applicatie ontwikkeld wordt. Bij ons betekent dat dat je van begin af aan een service manager moet laten meelopen, want die is bij ons verantwoordelijk voor het in beheer nemen. In de praktijk zie je vaak meerdere sprints voor één oplevering. Ik zie niet waarom we het incident proces dat we nu ook voor Exact gebruiken niet hiervoor zouden kunnen gebruiken, maar dat is dan geen agile beheer. Eigenlijk is agile beheer volgens mij hetzelfde als conventioneel beheer. De essentie van agile en sprints is beginnen terwijl de klant geen budget heeft en nog niet weet wat hij wil hebben. Er is een bepaalde behoefte en die ga je samen met de klant uitwerken. Dit is natuurlijk heel moeilijk, een project starten terwijl je niet weet wat het is en wat het gaat kosten. Het verschil tussen agile en conventioneel beheer is dat er bij conventioneel beheer een planning voor de ontwikkeling wordt gemaakt en aan het eind van die planning wordt het overgedragen aan beheer. Bij agile projecten kan het al tussentijds worden overgedragen aan beheer. Belangrijk is dan het betrekken van de service manager in het ontwikkeltraject. Daarnaast is plannen heel belangrijk, ontwikkelaars en beheerders kunnen namelijk dezelfde personen zijn, maar zitten niet niets te doen en te wachten totdat ze een beheertaak moeten gaan uitvoeren. Het verschil is dat je bij delivery gestructureerd met een project bezig bent en bij beheer niet, bij de laatste kun je zomaar een “urgent” melding krijgen waarbij we binnen 8 uur een oplossing moeten leveren. Daarnaast is het allemaal maatwerk en weet de ene ontwikkelaar niet wat de andere ontwikkelaar gemaakt heeft. Misschien is standaardisatie en uniformiteit van ontwikkeling dan een interessant aspect, want dat zou er moeten zijn om het te beheren.
Mijn idee is dat het traditionele beheerproces toereikend is in een agile/SaaS omgeving als je het maar flexibel inricht. Het probleem van een beheerproces dat kort cyclisch wordt ingezet is niet dat het gestandaardiseerde proces anders moet zijn, maar wel de capaciteit. Het draait ook om een mooie term. Toen we het “hosting” noemden kwam niemand naar onze seminars en nu we het “cloud” noemen zit het vol.
Je moet onderscheid maken tussen een shared en private omgeving. Bij private heb je natuurlijk een eigen omgeving ter beschikking binnen een omgeving, terwijl er bij shared gewoon virtuele machines draaien met nog meer klanten op één omgeving. Dit is een cruciaal verschil. Als je wilt updaten of aan een bepaalde vraag van de klant wilt voldoen zul je wel de samenhang moeten zien. In een private omgeving kun je het gewoon doorvoeren, want daar zit maar één man op. Het is hetzelfde verschil als tussen shared hosting en dedicated hosting bij een provider.
Bij de overdracht naar beheer hebben wij een eigen NAP (Nobel Aanpak Projecten). De fase “overdracht naar beheer” hierin heb ik samen met de bedenker van het NAP op basis van PRINCE2 uitgeschreven en daaraan een aantal configuratiedocumenten gekoppeld. Elke projectmanager moet mij een taakverzoek sturen voor de decharge van een project. Hier moet natuurlijk wel documentatie bij zitten. Als ik dat binnenkrijg vraag ik eerst aan de teamleider servicedesk of hij voldoende informatie heeft om decharge te verlenen. Als hij voldoende weet geef ik het project decharge. Vanaf dat moment is de projectmanager er van af en mag de klant ons bellen en hoeven ze niet meer de projectmanager te bellen. In een snelle cyclus van sprints en opleveringen moet deze documentatie wel op orde zijn om het aan ons over te dragen. Dit is niet nodig als je de consultant als 2^{e} lijn achter de hand hebt, maar een collega consultant weet het waarschijnlijk ook niet. De wrijving is dat je documentatie nodig hebt als je in de 1^{e} lijn zo ver mogelijk in dat 2^{e} lijns proces wilt komen. Dan móet je weten hoe het gebouwd is, maar daar is vaak geen tijd voor want de klant wil direct verder met de volgende iteratie (fase). Comments in het Mendix model gebruiken zou al voldoende kunnen zijn. Bij Mendix gaat het erom dat je snapt hoe het proces werkt en er functioneel uitziet. De techniek is minder belangrijk.
Dat capacity management en availability management is samengevoegd in het proces ICT operations management kan ik me voorstellen.
Scheduled jobs zijn het beste voorbeeld van ICT operations control. In een SaaS-omgeving is dit van toepassing, zeker je backups. De omgeving moet wel gebackupt worden en daar zijn processen voor. Ook binnen Mendix zul je backups moeten maken. Mendix werkt releasematig, dus je zult altijd een release terug moeten kunnen, dus die moet je ook bewaren.
De benoemde geautomatiseerde processen/taken hebben voornamelijk met de infrastructuur (IaaS) te maken, niet specifiek met Mendix. Automated scaling heeft een relatie met capacity management. Operational lifecycle management zegt mij niets. Workload management is meer load balancing. Ik herken geen gekke dingen in deze processen/taken. Misschien zou je UC’s (underpinning contract) er nog aan toe kunnen voegen, naast SLA’s, bijvoorbeeld in het voorbeeld dat ik eerder al gaf (wanneer een pakket van een andere leverancier input levert aan een pakket dat wij leveren). Bijvoorbeeld voor een aparte applicatie die de wisselkoers berekent en interfacet met Mendix. Als je dat namelijk in Mendix moet maken zou je elk uur een nieuwe release moeten draaien. Een underpinning contract heeft met de SLA te maken, maar de SLA is echt gefocust op de kwaliteit van de dienstverlening. De SLA beschrijft welke diensten je voor de klant uitvoert, is in wezen

een vertaling van de PDC. Een UC heeft te maken met onderliggende partijen. UC heeft met supplier management te maken.

Wellicht kun je beter het framework uit elkaar trekken en een apart framework relateren aan de applicatie en een ander framework aan de infra. Helemaal samenvoegen zou ook kunnen.

Change Management
De input voor het change management proces kan naast de beschreven bronnen ook komen vanuit het event management proces of het service (level) management proces natuurlijk als een service manager een vraag krijgt. Primair gezien zal het wel het vaakst uit incident en problem management komen.

Heel belangrijk is dat een RfC door de klant geaccordeerd moet worden, want die kost geld. Er komt in wezen een soort offerte achteraan. De klant vraagt een change aan, wij maken een RfC (helemaal webbased) en als die gesubmit wordt komt hij bij de klant in zijn workflow in zijn customer portal. De klant kan dan in de customer portal kiezen voor deny, approve of toevoegingen geven. Er staat hier een bedrag onderin, dus als hij het approved heb je meteen ook de goedkeuring. Denied hij, dan geeft hij aan waarom en passen wij het aan of we doen het niet. Zo simpel is dat proces, maar het is wel belangrijk. Als je namelijk een RfC gaat uitvoeren en het staat niet in je contract, dan kost het geld. Zelfs als het in het contract zit kost het meestal ook geld. Standaard changes kunnen in bulk afgekocht worden door klanten.
Er zal bij een change altijd ergens een prijs moeten zitten en daarnaast heb je altijd iemand nodig die dat moet gaan doen, dus die moet gepland worden. Consultants zitten namelijk niet te wachten totdat er een RfC komt, die zijn ook bezig met ontwikkeling en ook met beheer en dat moet in de agenda ingepland worden.
Ookal is het Mendix, je zult toch je resources moeten plannen. Het zal in Mendix wel allemaal sneller gaan, maar je zult toch een poppetje moeten hebben die het doet.
Wij werken altijd met een planning en zeker dat stuk is ook in een Mendixomgeving belangrijk. De jongens worden allemaal ingepland, óf op een project óf voor support. Je ontkomt er niet aan om bepaalde tijd in te ruimen.

Categorisatie/prioritering is belangrijk omdat het best kan zijn dat een urgent change versneld doorgevoerd moet worden. Dan zul je er sneller mensen op moeten zetten en spreek je niet over 2 dagen, maar over 2 uur bijvoorbeeld. Vaak ga je met incident management proberen die omgeving in de lucht te krijgen en probeer je de workaround toe te passen, maar soms kan dat niet en dan zul je toch heel snel een change moeten gaan doorvoeren. Toevallig hebben we dat laatst gehad. Er was toen binnen 2 uur een RfC geschreven en een uur later was de RfC doorgevoerd. De hele doorrekening ervan hebben we toen later gedaan. Dat is een aparte procedure voor urgente zaken, dus dat is volgens het traditionele proces.

Ik mis de link naar release management en de CAB. ITIL versie 3 noemt de CAB volgens mij anders. De CAB beslist niet alleen over kostbare changes qua prijs en tijd, maar ook of het niet conflicteert met de huidige functionaliteit. Het kan best zijn dat een bepaalde change doorgevoerd moet worden die een gevolg heeft voor andere dingen. Dan moet je een analyse uitvoeren. Deze analyse zit in de activiteit “Impact and resources”, want daarvoor heb je de gegevens van configuration management nodig.

Vanuit de klant moet uiteindelijk altijd akkoord worden gegeven voor een wijziging op zijn productieomgeving. Een change ga je waarschijnlijk altijd eerst testen in je testomgeving en als het daar werkt ga je naar productie, maar een productietest is nooit gelijk. Dit kan hem zelfs zitten in een klein verschil in de processor. Je krijgt nooit een systeem gelijk.

Bij “does it work” kun je er ook achter komen dat er een probleem zit in je classificatie. Er kan vanuit de inschatting een fout zijn gemaakt in de oplossingsrichting. In het proces zou ik dus altijd de mogelijkheid openhouden om terug te kunnen naar die fase.
“Build” en “Model” kun je eigenlijk zien als gelijk. Of het stukje “transform” klopt weet ik niet. “Transform” is eigenlijk het vertalen van het model, wat je in het traditionele proces bij “implement” doet. De juiste volgorde is: bouwen, testen, kijken of het werkt, implementeren. Testen is ook wel kijken of het werkt, maar dat zijn vaak deeltesten en als je gaat implementeren krijg je de hele keten en dat is toch weer een ander soort test.

Een evaluatie is altijd belangrijk. Als je de tijd ervoor hebt zou evaluatie van een incident ook al heel veel kunnen oplossen, maar daar is de tijd gewoon niet voor. Daar moet je realistisch in zijn en daarom zit evaluatie

ook niet in incident management en is problem management juist ontstaan, om daar wel tijd voor te maken. Problem management gaat over het analyseren van incidenten, herkennen, uitzoeken en de oplossing testen en doorvoeren/klant op de hoogte brengen.
Wat betreft de evaluatie is het met name interessant als de “does it work” een “No” is geweest en je komt na het opnieuw doorlopen van het proces weer terug bij dit keuzepunt. Het is ergens niet goed gegaan, dus bij de evaluatie komen de lessons learnt naar voren. Als het proces in één keer goed wordt doorlopen heb je er waarschijnlijk weinig aan, want dan is het gewoon routine, maar als je de “No” één of meerdere keren moet gebruiken zou ik de evaluate uitvoeren. Interessant is de vraag: “Waarom wijken de TO en FO af?”.
Iedereen zegt te willen leren, in de praktijk doet bijna niemand het. Maar als je wilt leren moet je het vastleggen en ook uitdragen. Je past je proces er dan ook op aan. In de praktijk doen wij deze evaluatie niet. Bedrijven verkopen deze activiteit voornamelijk als extra omzet aan klanten en ik geef ze ook wel gelijk, want het is wel belangrijk. Maar wat doet de klant uiteindelijk met het evaluatierapport? Ik heb nog bij geen enkele klant gezien dat ze daar ook echt effectief iets mee deden.

Het belang van configuration management in het change proces ligt er helemaal aan hoe configuration management is ingericht. Dat doet iedereen op zijn eigen manier. Met name als je de aanpassing doorvoert is configuration management van belang. Je hebt tools voor configuration management, maar ook de diepte leg je vast. Qua diepte leggen wij nu vast: het pakket en de versie. Je zou ook geïmplementeerde modules of patches bij kunnen houden, maar dat doen wij niet.

Ik denk dat change management redelijk één-op-één is (resp. traditioneel vs. SaaS).

Configuration Management
Wat betreft configuratie management neem ik even applicatiebeheer, hoe het in Naarden (onze infra-tak) zit weet ik niet.

Voor Mendix configuration management zou je de Mendixversie en de release vast kunnen leggen. Bijvoorbeeld de platformversie en je zou een versienummer kunnen koppelen aan wat je gebouwd hebt. Die leg je dan ook vast. Ik heb me niet verdiept in wat je nog meer vast zou moeten leggen. Voor Exact Synergy en Globe leggen wij gewoon de versie vast en daarnaast ons asset management. Onze assets zijn onder andere BI (QlikView). Voor ons is het belangrijkste om te weten wat (welke producten) de klant heeft en welke versie. Het feit dat ik weet dat de klant Mendix heeft is al belangrijk. Dit is wat anders dan CRM, maar ons CMDB hangt wel onder de klantkaart in ons CRM systeem. Als we straks naar een ander ISM systeem gaan gaan we dat anders doen, maar de link moet er wel altijd zijn want als de klant belt kunnen we direct een link maken naar ons CMDB. We noemen dat de relatiematrix.
Voor Mendix houden we alleen bij dat het Mendix is. We houden geen versienummers of applicatiecomponenten bij. Dit is voor ons op dit moment het belangrijkste. Het is niet het allerbelangrijkste om dit nu in te richten voor ons, want alle informatie staat in het Mendixsysteem zelf en die consultants hebben heel gauw in de gaten wat voor versie het is. Een CMDB inrichten zonder dat dit consequent wordt bijgewerkt vanuit het change proces is kansloos en alle consultants de wijzigingen die ze doen bij klanten laten aanpassen op de relatiekaart zie ik niet zo snel gebeuren. De cultuur en de manier van werken is anders. We weten heel vaak gewoon wat er bij de klant staat, dus dan is apart vastleggen niet nodig. Klanten waarvan je relatief weinig hoort zul je dan wel eens een vraag moeten stellen. Bovendien kun je het verleden zien door te scrollen door de incidenten.

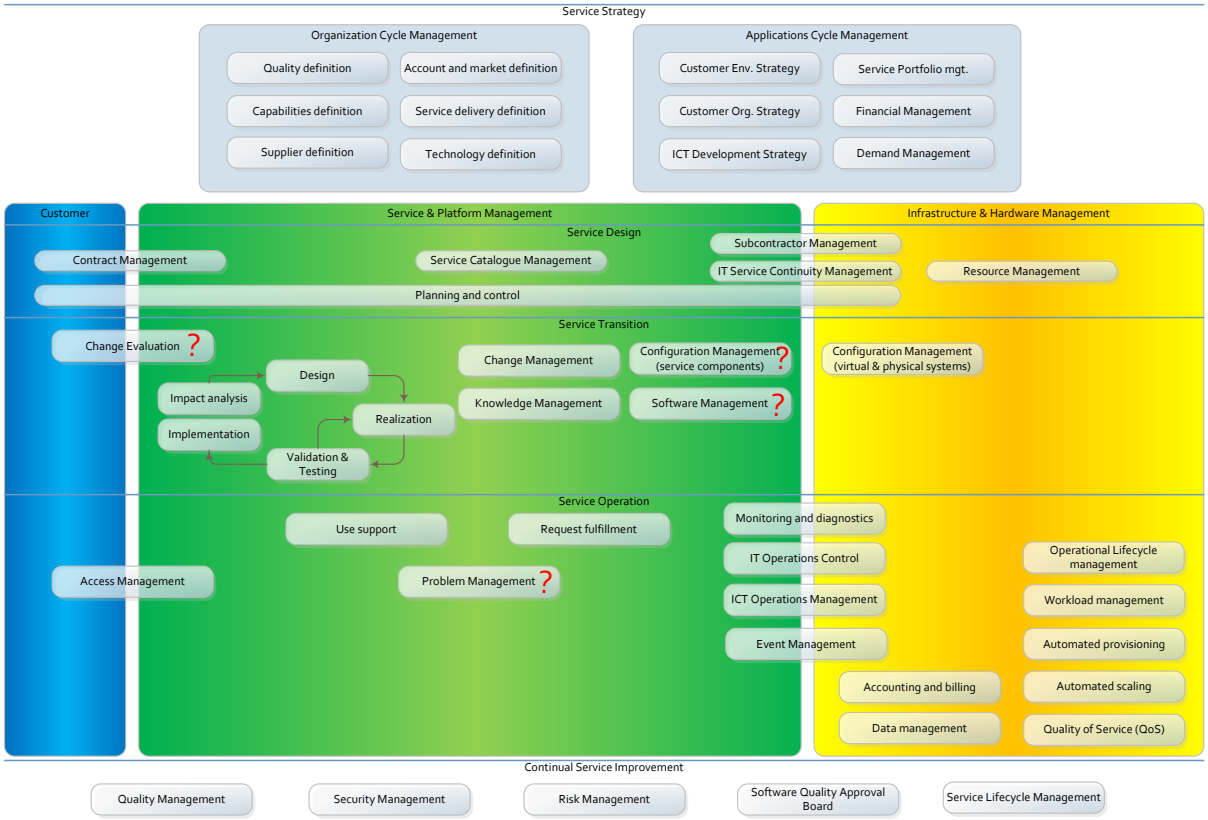
In de praktijk zit je continu op het snijvlak van P&L (Profit and Loss; mijn kostenplaats op de balans) en kwaliteit. Ik moet dus geld verdienen en daarnaast de kwaliteit borgen. Het is een continue afweging of ik de uren besteed aan het doorvoeren van verbeteringen, aan het incident proces of eventueel het configuration proces.

Define configuration plan: we hebben eenmalig bepaald wat we vastleggen. Dat is de relatiematrix en ons CMDB.
Het datamodel daarachter hebben we ook gedefinieerd; welke velden zijn er die gevuld kunnen worden, niet elke klant heeft dezelfde set aan zaken.
Naming conventions hebben we niet.

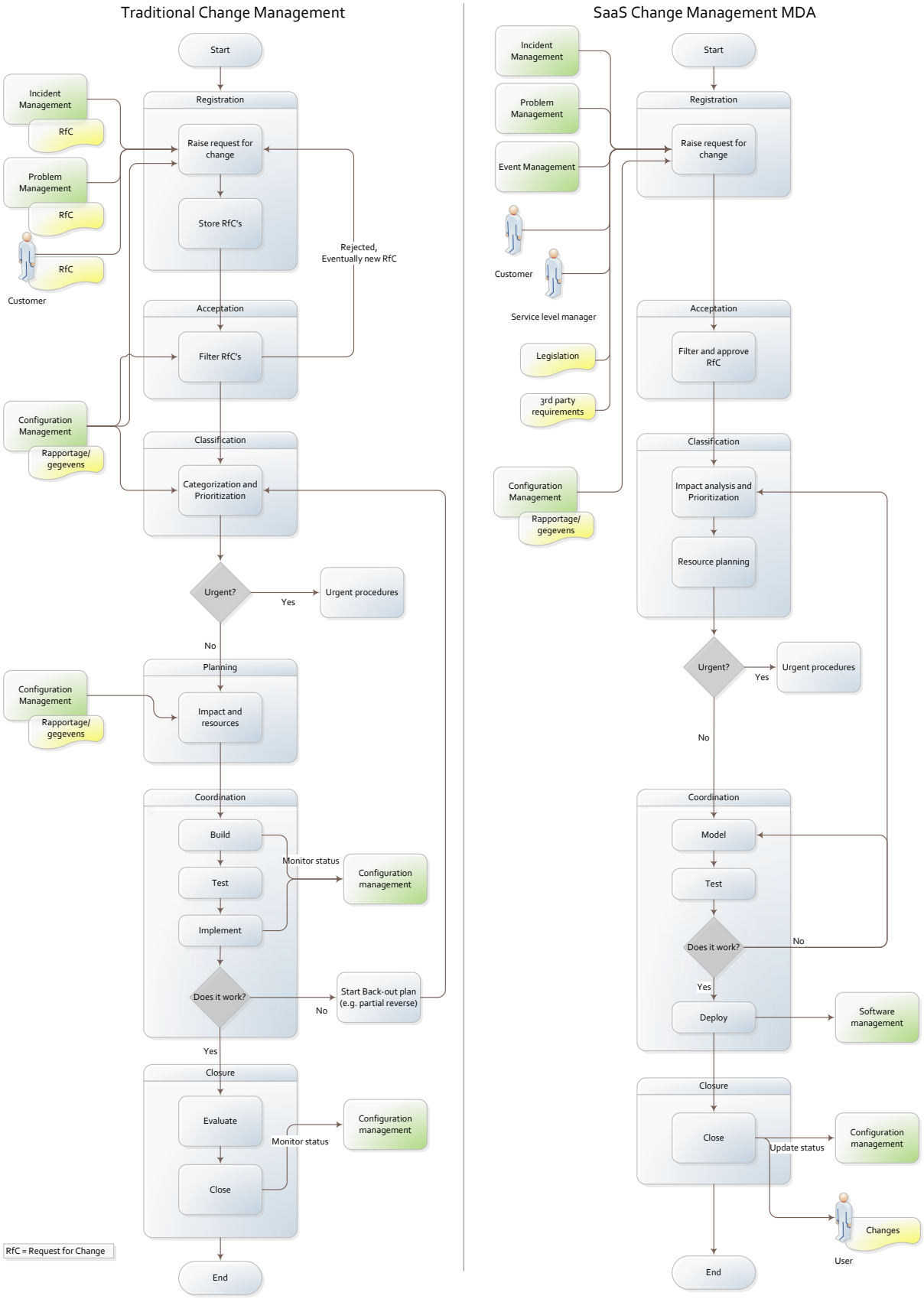
Registreren van CI's doen we op het moment dat we het weten. Het zou echter kunnen dat wijzigingen niet altijd worden doorgevoerd vanuit het change proces. We doen updates op het moment dat we een update krijgen van een consultant of een klant.
Disclosure van CI's is mogelijk; je kunt dit gewoon uit het CRM systeem halen.
Monitoring en audits zijn volgens mij niet van toepassing op het configuration management.

APPENDIX I: THE SECOND CONCEPT OF THE THEORY

THE SAAS MANAGEMENT FRAMEWORK

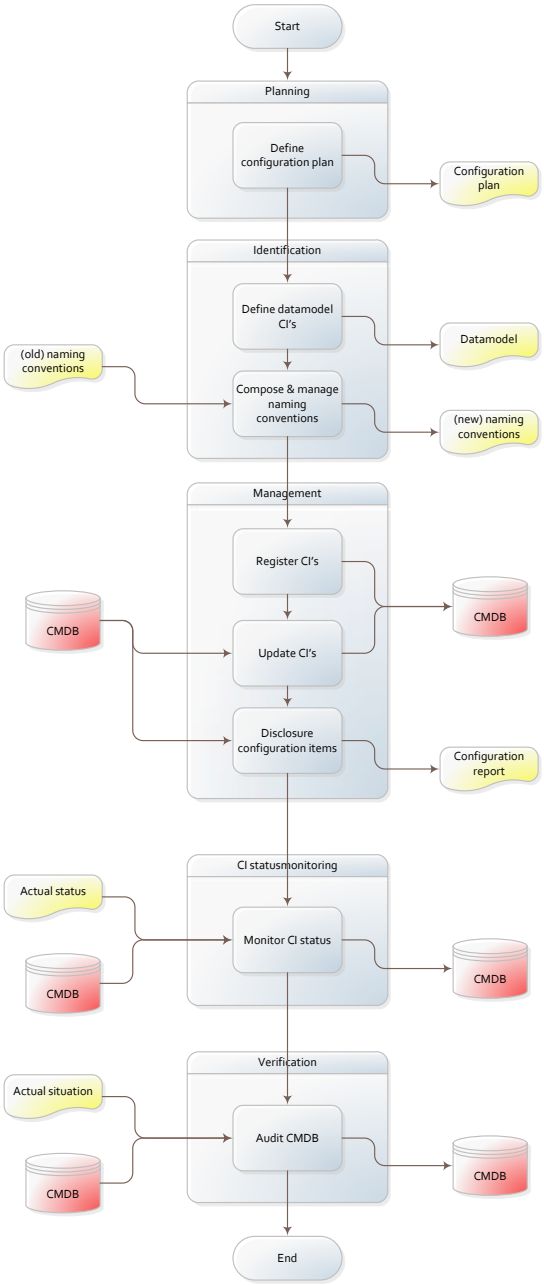


THE CHANGE MANAGEMENT PROCESS

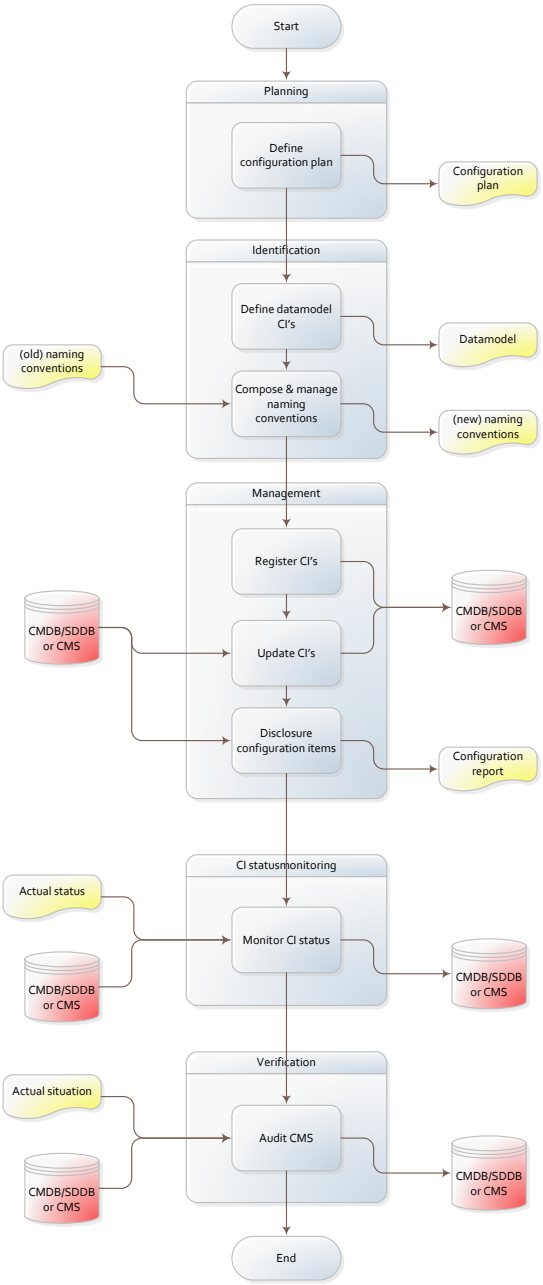


THE CONFIGURATION MANAGEMENT PROCESS

Traditional Configuration Management



SaaS Configuration Management



CI = Configuration Item
CMDB = Configuration Management Database
SDDB = Service Delivery Database
CMS = Configuration Management System

APPENDIX J: VALIDATION REVIEW COMPANY C

To perform the framework validation case study, an interview has taken place. The introduction text of the case study refers to external sources which have additionally been used to prepare the case study and verify some of the subjects that were mentioned in the interviews.

Interview		
Code	**Position**	**Date**
C-I1	IT Director/Manager Business IT & Manager Projects	16-11-2012

Table 7: Interviews held in the validation within company C

The interview reports are not literal transcriptions of the interviews, but mentioned aspects are partly summarized and shuffled to present a more logical story. As much information as possible is included in the interview reports to not accidentally overlook important aspects that may have gotten lost when summarizing. The digital recordings of the interviews as well as the mentioned documentation are available for inspection with the author.

INTERVIEW REPORT C-I1

Interviewee 1 position: Manager Projects
Interviewee 2 position: IT Director & Manager Business IT
Interview date: 16-11-2012
Interview duration: 55 minutes

Introduction
Interviewee 1 is aan het woord.

De IT afdeling bij PostNL bestaat uit 4 operationele onderdelen:
- Run SAP
- Run IT (niet SAP)
- Service IT (= aanbodgedreven diensten zoals bijvoorbeeld telefoons, computers en officesystemen)
- Business IT, opgesplitst in de volgende 4 takken:
    - Architectuur
    - Informatiemanagement
    - Business requirement
    - Projecten

Interviewee 2 is zowel manager van het “Business IT” onderdeel als IT directeur van PostNL. Het “Business IT” onderdeel zit eigenlijk het meest kort op de business en is de regievoerder van het hele IT veld/het hele portfolio van de post.
Wij zitten midden in een transitie dat we veel systemen naar de cloud gaan trekken. Toen jij het had over jouw onderzoek ging bij mij een lichtje branden: “misschien is dat wel handig voor ons”. Interviewee 2 zit daar middenin. Het programma wordt getrokken door een programmamanager en een stuurgroep waar interviewee 2 in zit en ik zit daar niet in. Daarom was het handig om ook met interviewee 2 dit gesprek te voeren.

Interviewee 2 is aan het woord.

Verandermanagement is mijn leven. Met de term “ITIL voor SaaS” ben ik het al niet eens. Over het framework wil ik conceptueel het volgende zeggen: SaaS staat voor Software-as-a-Service, dus dan is er iets wat al functioneert en je wordt gebruiker. ITIL komt uit de oertijd van IT. Dat zit meer in de infrastructuur-wereld en probeert al jaren omhoog te kruipen naar de proces/bedrijfsproceswereld. In mijn ogen is dat nog nooit gelukt. In Nederland hebben we de ASL en BiSL beweging wat eigenlijk een laag op ITIL is om ITIL voor niet-IT’ers werkbaar te maken, eerst voor applicaties en toen voor bedrijven. Als je naar SaaS kijkt zou BiSL eerder het uitgangspunt moeten zijn dan ITIL, omdat BiSL gaat over het brug slaan tussen wat het bedrijfsproces nodig heeft om te functioneren. Dat wordt vertaald in een vraag voor ASL. Omdat ASL niet meer hoeft, want dat is er

al in de vorm van SaaS, krijg je dat er tussen BiSL en wat vroeger ASL heette ten opzichte van SaaS een soort brug geslagen moet worden die niet zozeer over applicatiebeheer gaat, maar veel meer over het functioneel beheren van de SaaS (een soort “admin-rol”). Dat is steeds minder technisch en steeds meer een soort super-supergebruiker. Dat is wat je ziet. ITIL als uitgangspunt van jouw studie vind ik daarom een question mark.

Infrastructuur, service en platform zijn dingen die allemaal IN de SaaS onderwater zitten, dus eigenlijk wil je dat allemaal niet weten en wil je er sowieso niks op gaan beheren, want dat doet iemand anders.

Mendix is eigenlijk een platform waarmee je snel software kan genereren. Dat is meer dan een PaaS en minder dan een SaaS. Er zitten al functieblokken in. In de methodiek zit al functionaliteit of het genereren van functionaliteit. Bij een PaaS moet je de functionaliteit nog bouwen/ontwikkelen en neerzetten, terwijl het er hier eigenlijk al in zit danwel gegenereerd wordt op basis van andere input. Het is een ander abstractieniveau. Op .NET basis heb ik eigenlijk alleen maar een executie-omgeving van een .NET applicatie, terwijl als ik een Mendixplatform heb, dan heb ik eigenlijk een softwaregeneratie-engine die ook runtime werkt. Dat is anders, het zit niet op hetzelfde niveau als een compiler omdat een compiler een executable oplevert en Mendix een model oplevert dat met een engine geëxecuteerd wordt. Het is misschien maar een gradueel verschil, maar bij Mendix heb ik gewoon geen programmeur nodig, maar business modellers. Als ik .NET bouw heb ik te maken met een Functioneel Ontwerp, Technisch Ontwerp, hackers, testers. Dat is gewoon een heel andere wereld. Zeker als je het hebt over hoe je het moet aansturen.

SaaS management framework
Contract management: het management van de SLA met de klant blijft en wordt steeds belangrijker. Het worden steeds meer functionele gebruiksaspecten en continuïteitsaspecten van een bedrijfsproces in een organisatie in plaats van “doet ie het of doet ie het niet”. In de oude wereld heb je te maken met de needs en “hoelang duurt het voordat hij het weer doet”, maar hier heb je ook veel meer te maken met “doet ie het goed”, “doet ie het snel genoeg”, “wat gebeurt er als ie het niet doet”, “hoe weet ik het überhaupt als ie het niet doet”. Het belang en ook de afhankelijkheid van dat dit goed geregeld is en het ook goed ingeregeld is in de organisatie van de SaaS provider is veel groter dan in het verleden omdat ITIL natuurlijk voortkwam uit interne IT processen en nu zijn het externe IT processen van één of vaak meerdere service providers gestapeld. Je zit aan de buitenkant en dan moet je dus regelen dat het daarbinnen geregeld is, maar je kan niet naar binnen kijken want het is niet van jou. Je moet dus nadenken over hoe je zeker krijgt wat je nodig hebt. Dit proces is iets wat bij zowel de klant als de service provider ligt.
Vroeger sloot inkoop het contract, dan legde je het op de plank en deed je er niets meer mee. Er was dan een service manager die af en toe op bezoek kwam vanuit de leverancier. Nu krijg je dat je veel meer een hardere handshake moet hebben tussen je eigen bedrijfsvoering en de bedrijfsvoering van de SaaS-leverancier om te zorgen dat je samen iets laat functioneren. Traditioneel had je een service manager van de leverancier en een service level manager van de klant/business kant (als die er al was). De service level manager was vaak iemand van de gebruikersorganisatie terwijl los daarvan inkoop contractmanagement was. Het afspreken van een contract en daarna borgen dat je dat ook krijgt zie je steeds meer gaan integreren, omdat de frequentie waarmee en het detailniveau waarop het gebeurt veel groter wordt. Het wordt veel ingewikkelder om af te dwingen wat je nodig hebt en dat te bewaken. Vroeger was het één keer per jaar of één keer per 3 jaar een contract sluiten, inkoop regelde dat en daarna was het klaar. De bewaking wordt intensiever omdat het nu direct inpact heeft op hoe je bedrijf functioneert. Als vroeger je computer het niet deed regelde de interne IT afdeling dat wel. Je schuift als het ware in de keten op, waardoor je belangen veel groter worden. Je integreert ook allerlei functies die je vroeger intern had ineens in één contract, in één afspraak die je moet bewaken. Wij hadden hier applicatie service en infrastructuur service als afdeling. De applicatieserviceafdeling regelde dat de computer het deed, de business wist er helemaal niets van dat dat geregeld moest worden. Nu zitten ze helemaal aan de beginkant en alles wat daaronder zit moet geregeld zijn en zij moeten vooral sturen dat ze krijgen wat ze nodig hebben. Het wordt allemaal veel intensiever, ook omdat de afhankelijkheid groter wordt.

Service catalogue management is eigenlijk de verbastering van configuration management, maar dan is dit wat er onderwater zit verpakt in de services die je afneemt. Als dit proces niet is ingericht kan dat een volwassenheidsprobleem zijn in de markt, maar het kan ook dat het er gewoon nooit van komt. Wij hebben een project met de naam “Fusion” waarin we hele werkplekken en office-omgeving aanbieden. Daarin hebben we wel een service catalogus die zelfs de gebruiker online kan oproepen en services kan aanvragen die dan automatisch geleverd worden. Daar hangt dan ook weer de rekening aan die we krijgen. In zo’n context

functioneert dat goed. Voor de leverancier van Fusion zit daar configuration management op, want als wij een werkplek of account geleverd krijgen gaat daar een rekening voor lopen.

Subcontractor management klopt op de plek in het framework waar je hem hebt neergezet. Dit proces is vooral belangrijk voor de SaaS-leverancier in zijn compliancy-bewijslasten. Wij hebben als IT afdeling intern allerlei compliancy regels; wij moeten bewaken dat ongeautoriseerde mensen niet aan de applicatie kunnen komen en dat soort toestanden. Wat je ziet is dat die bewijsvoering voor een deel verschuift naar de SaaS-leverancier. Op het moment dat hij subcontractors heeft moet hij ook borgen dat die voldoen aan onze compliancy-bewijslasten. Hij moet bewijzen dat er niemand in onze applicatie is ingelogd en dat niemand bij de data heeft kunnen komen, ook niet als iemand systeembeheerder was van een systeem. Dat moet hij bewijzen tot op CEO niveau, dat moeten ze tekenen.
Wij geloven niet dat je compliancy apart moet organiseren, maar dat je het geïntegreerd in je processen moet organiseren. Het compliance-proces is hooguit het proces waarin vastgesteld wordt dat het functioneert. Wij integreren onze compliancy helemaal in onze processen, dus als wij moeten bewijzen dat er niemand met zijn vingers aan het systeem heeft gezeten hebben wij daar monitoring op. Er wordt een incident geregistreerd als iemand foutief inlogt of het gehackt heeft, of om technische redenen bij de applicatie moest. Wij hebben bijvoorbeeld een “rode envelop-procedure”; als een systeembeheerder in de productieomgeving moet inloggen omdat daar iets fout is en gerepareerd moet worden wordt dat vastgelegd. Alles wat hij daar doet wordt gemonitord en later wordt er een rapport geschreven waarin bewezen wordt dat hij wel zijn werk gedaan heeft maar niet in de data heeft gekeken. Compliancy moet je integreren in je processen, want anders kun je het nooit achteraf bewijzen. Als je niet bewaakt wat iemand in de productieomgeving doet kun je niet bewijzen dat hij niet aan de data heeft gezeten.

IT service continuity management is een gestapeld proces dat zowel speelt bij de infrastructuurleverancier als bij de service/platformleverancier. Bij ons zit dit proces ook tussen de klant en de service/platformleverancier. Ik moet namelijk ook mijn continuiteit borgen. Het gaat over: “hoe kan ik opereren als er een verstoring is bij de SaaS”. Eigenlijk is het proces belangrijk voor alle drie de partijen.

Planning and control. Als je er vanuit gaat dat een SaaS-situatie een aanbod-gedreven situatie is, dus de SaaS-leverancier heeft iets aangeboden aan de markt en hij doet dat breder dan alleen voor mij als klant en hij heeft zijn eigen levenscyclus voor dat product dan is het planproces van de SaaS meer losgekoppeld van mijn planproces. Voor mij is het een gegeven dat die SaaS er is. Dat er dingen in die SaaS komen volgens de plancyclus van de SaaSleverancier hoor ik van hem. Misschien kan ik invloed uitoefenen, maar dat is minder procesmatig en veel meer relationeel, dat er bepaalde functionaliteiten in komen. Wanneer het er in komt en of het er dan precies in komt zoals ik het wil hebben is toch losser gekoppeld, want dat is zijn SaaSproduct marketingdevelopment. Dat betekent dat mijn planning and control proces autonoom is en daarin heb ik mogelijk een aanbod van functionaliteit uit de SaaS die ik daarin kan plannen, maar voor of het er is en wanneer het er is ben ik afhankelijk van wat hij aanbiedt. Ook tussen een SaaS/PaaS en een IaaS is een identieke vorm van samenwerking. Ook een IaaS heeft een aanbodgedreven productaanbod. Dat is een gegeven en dat kun je gebruiken of niet. Als je het niet kunt gebruiken moet je iets anders verzinnen. Die planning is niet echt gekoppeld, er zit een schotje tussen. Als de PaaS/IaaS iets nodig heeft wat de IaaS niet levert ga je naar een ander zoeken, of je kunt niets leveren aan je klant. Ik zou er twee schotjes tussen tekenen.

Resource management: als het gaat over capaciteit van systemen dan zit hij inderdaad bij de infrastructuur- en hardwareleverancier, als je het hebt over resource management in termen van personeel dan zit het proces natuurlijk integraal. In ITIL is resource management volgens mij capaciteitsmanagement. Nou ken ik ITIL versie 3 niet goed, ik ben wat ouder.

Change management. Wij hebben Salesforce als SaaS en die neem ik even als voorbeeld. Salesforce levert gewoon iedere drie/vier maanden een nieuwe versie op in hun SaaS. Die krijg ik gewoon. Ik krijg vier weken van tevoren een aankondiging en ik krijg hem dan in mijn testomgevingen als ik die heb, zodat ik even kan kijken of alles het nog doet, maar ik kan niet “nee” zeggen ofzo. Dus het change management in de traditionele zin van “ik heb een wens, ga een impactanalyse doen, enz. enz. en op een gegeven moment wordt er iets gemaakt, opgeleverd, dat kan ik testen en zeggen dat het goed is en dan wordt het geïnstalleerd in mijn productieomgeving”, dat is er helemaal niet meer. Ik krijg het gewoon. Ik kan het dus wel testen van tevoren, maar als ik me aan de regels gehouden heb is dat veel meer een validatie en als ik echt een bug vind gaan ze die natuurlijk ook oplossen, want dan heeft de rest van de wereld er ook last van. Als ik me aan de regels heb

gehouden van Salesforce dan is er een soort van upwards-compatible model wat betekent dat alles bij mij nog zou moeten functioneren, plus dat ik er functies bijgekregen heb die ik dan eventueel ook aan kan zetten.
Het traditionele change proces wordt voor de klant dus heel anders. Bij Salesforce zullen ze een changeproces hebben. Ik als klant heb ook een changeproces, maar dat gaat veel meer van “er is een nieuwe release van Salesforce met nieuwe functionaliteit”. Dat is meer aanbodgedreven. “heeft het zin dat ik die ga gebruiken?” is voor mij de vraag, maar daar hoef ik niet een IT change proces voor te doen, dat is veel meer een business changeproces. Je ziet dat een change vroeger heel erg IT-georiënteerd was, voor wat betreft het maken van de wijziging althans, en je daarnaast nog een change aan de businesskant had. Je ziet dat het hele concept van change nu verandert: aan de businesskant wordt het veel belangrijker, daar wordt het veel meer “ik heb nu functionaliteit gekregen van de SaaS. Ga ik die gebruiken en wat is het effect daarvan?”. We zien veel meer optimalisatie in het change managementproces aan de businesskant.

Change evaluation zoals jij het beschrijft is in feite de marketingafdeling van in mijn voorbeeld Salesforce. Wij hebben ook met Salesforce ineens allerlei mensen die we niet kennen als rol. We hebben een customer succession manager, dat is iemand die ons eigenlijk achter de broek zit om te onderzoeken of de functionaliteit in Salesforce bij ons van toepassing is c.q. beter gebruikt kan worden. Dit is een hele nieuwe functie. Ik heb ook een customer success manager: “zijn wij als Salesforce in staat om met behulp van Salesforce nieuwe functionaliteit te genereren in de organisatie waardoor we beter functioneren”. Er komt een enorme focus op het benutten van de mogelijkheden van Salesforce, want dat is hun levenslijn. Hoe meer ik ervan ga gebruiken, hoe meer value voor money het voor mij heeft en hoe eerder ik mijn abonnement niet op ga zeggen want de SaaS is natuurlijk gewoon een abonnementssysteem. Ik kan gewoon Salesforce opzeggen, de rekening niet meer betalen en het weggooien. Daar zit veel meer focus op, dus dat proces zou ik positioneren tussen klant en SaaS leverancier vanuit marketingoptiek. Dit proces zou je het “customer succession proces” kunnen noemen. Dat sluit aan bij een changeproces aan de businesskant; Je moet veel meer aandacht besteden aan het benutten van de mogelijkheden van de SaaS. Je komt dus veel meer in een aanbodgedreven manier van IT bedrijven dan in een vraaggestuurde. De rollen die nu in het framework staan zijn allemaal reactief, vraaggestuurd; het bedrijf heeft iets nodig, dan ga ik het maken/bestellen en laten werken en als het niet goed functioneert kan ik iemand op zijn hoofd timmeren met toestanden als service level management, terwijl de wereld in SaaS veel meer een aanbodgedreven situatie wordt en je dus moet kijken of je het effectief gebruikt/beter kunt benutten of als ik iets verander in mijn organisatie daar uit het SaaS-aanbod iets bij kan gebruiken. Je krijgt een heel ander spelletje. De andere kant op zie je dat Salesforce met een heleboel klanten contact heeft, dat is anders dan één-op-één. Er gebeurt vanalles in de wereld, dat tellen zij op en dat leidt tot een nieuwe release. Het is dus nooit “ik vraag iets” maar altijd “de markt vraagt iets” en ik ben onderdeel van de markt dus ik heb het misschien ook gevraagd, maar ik alleen ben niet de markt. Ze gaan het voor mij alleen niet maken, ze gaan het wel maken als het een generieke requirement is voor een grote klantgroep.
In mijn voorbeeld heeft Salesforce natuurlijk intern een change management proces, maar daar zit dan een bruggetje boven naar een proces dat generiek de requirements bepaalt, op basis van wat de markt vraagt. Dat wordt dan een nieuwe release of change. Wij als klant hebben een ander soort van change management proces nodig dat veel meer plaatsvindt op basis van het aanbod van Salesforce dan op basis van onze specifieke vragen.

Configuration management is zowel op applicatie- als op hardwareniveau van toepassing. Ook voor Mendix. Mendix is eigenlijk een serviceverzameling die op een gegeven moment in een workflow/bedrijfsproces aan elkaar geknoopt worden en tot een werkend geheel gemaakt worden. Salesforce is eigenlijk een ballenbak van functionaliteiten die je configureert, aan elkaar knoopt en dan is het ineens een applicatie. Dat is ook het bestaansrecht, want hoe meer zij services kunnen hergebruiken, hoe groter hun aanbod is. Binnen Salesforce zijn dat configureerbare eenheden binnen de standaardapplicatie. Daar heb ik bijvoorbeeld een object “cases” (een gebeurtenis) die ik kan configureren en in een workflow kan hangen en dus configureren wie daar wat mee moet doen en wat er gebeurt als hij dat niet doet. De configuratie hoe een proces werkt, een stukje inrichting, doet de klant. Dat zou ik configuration management in combinatie met de servicecatalogus noemen. Het is in feite: welke functionaliteit wordt enabled voor gebruik door de klant. Bij ons is dat de Salesforce configurator als voorbeeld. Bij Mendix is dat de factory die wij gekocht hebben, die dat gaan doen. Eigenlijk is het gewoon het aanzetten van de service, zodat de klant het kan gaan gebruiken.

Software management heette vroeger “deployment and distribution”. Het is volgens mij helemaal niet zo dat je in de cloud niet distribueert. In sommige soorten clouds is dit net zoals vroeger. Hoe meer het van vroeger is, hoe eerder het ook niet valt onder de definitie van cloud. Het is dan eigenlijk gewoon oude wijn in nieuwe

zakken. Het hangt er vanaf wat er onderwater in de cloud zit. Als hij niet generiek is, niet multi-tenant, dus niet door alle klanten gezamenlijk gebruikt dan is het al gauw eigenlijk gewoon een ouderwetse managed service-omgeving waarin iets specifiek voor jou neergezet is en dat wordt dan door iemand geserved. Ik ken iemand die zegt: “je moet vragen wat één user kost”. Als ze dan een ingewikkeld verhaal houden weet je dat het geen cloud is. Als je niet één user kunt kopen, maar je moet altijd de rollen kopen en ze moeten vanalles voor je doen voordat je het kan gebruiken, dan is het geen cloud. Dan is het gewoon maatwerk en hebben ze misschien wel generieke services die ze hergebruiken, maar dat doen wij hier als IT afdeling intern al jaren. Ik koop bijvoorbeeld ook niet een computer specifiek voor één klant. We hebben alles gevirtualiseerd, dus we maken een instance, we monitoren de capaciteit en op het moment dat het te klein is voegen we wat geheugen en I/O cycles toe en dan wordt hij sneller. Hij krijgt dan een groter aandeel van de kosten voor zijn rekening. Ik heb ook gewoon een prijslijst. Het criterium voor cloud is dus dat je het als klant aan- en uit kunt zetten.

Knowledge management wordt gebruikt om degene die de services gaat gebruiken uit te leggen waar het voor bedoeld is en hoe het moet functioneren. Het heeft met de klant te maken omdat het eigenlijk de helpfile is van de service catalogus. Een incidentensysteem wordt een knowledge systeem op het moment dat je er in registreert wat de oplossing was van het incident en dat er dan een “wat doe je als”-vraag gesteld kan worden. Dus in het geval van services: als ik klachtenafhandeling wil configureren in Salesforce, hoe doe ik dat dan? Dan zou er een soort van guidance moeten zijn hoe je dat inricht. Dat is met SaaS-oplossingen heel belangrijk: self-service van toepassingen te krijgt. Anders moet de SaaS-leverancier iedereen die het wil gebruiken helpen om het te leren gebruiken en dan gaat hij zijn volume niet halen. Bovendien zit je dan weer in de oude situatie dat je techneuten nodig hebt om de service te kunnen gaan gebruiken. Je kunt het proces niet doortrekken van customer naar IaaS provider. Ik zie hem aan de voorkant van de service dus daar waar je op de brug zit van gebruik en leveren. Het is belangrijk in het self-service concept van SaaS. De SaaS provider moet middelen introduceren, zeg maar een self-service concept rondom zijn SaaS.
Traditioneel heb je in de IT markt met partners te maken, bijvoorbeeld SAP-partners die SAP komen implementeren. Dat is dan tweederde van je kosten, daar moet je natuurlijk van af. Het moet dus zo gemaakt worden dat je als eindgebruiker met het knowledge systeem het systeem aan kan zetten. SAP is heel ver met het bouwen van een cloud. Daar heb je een soort van “superwizard” om SAP te configureren, terwijl je daar vroeger een batterij consultants voor nodig had.

De ontwikkelcyclus staat logisch in de “service transition” fase van de serviceleverancier.

Access management. Wij willen niet dat de service provider in ons systeem zit. Vanuit compliancy mag dat helemaal niet. Wij willen niet dat iemand van de SaaS-provider of van de infrastructuurprovider kan inloggen in onze productieomgeving. Als het om technische redenen wel gebeurt willen we dat weten en willen we dus ook kunnen vaststellen wat hij gedaan heeft. De SaaS-leverancier levert de omgeving en het vermogen om gebruikers en rollen te definiëren, maar wij doen het. De leverancier heeft ook een rol, maar wij moeten toestaan dat hij ergens in komt. Access management doen wij ook met een andere provider dan Salesforce. Wij hebben een identity access management systeem dat automatisch gebruikers creëert in Salesforce. In Salesforce hebben we alleen gebruikersprofielen gedefinieerd. Als we dan een gebruiker aanmaken is automatisch geregeld wat hij mag. Als de gebruiker uit dienst gaat gooien we hem er automatisch uit. Salesforce weet daar allemaal niks van, dat is gewoon een interface.
Monitoren is ook voor mij belangrijk. Ook mijn IT-afdeling mag niet inloggen op de productieomgeving. Er zijn geen IT gebruikers bevoegd om in te loggen. Er zijn wel gebruikersrollen voor IT’ers. Op het moment dat wij om technische redenen moeten inloggen op de productieomgeving wordt dat met de rode envelop-procedure geregistreerd, ook aan de businesskant dus ze weten dat we dat dan doen. Vervolgens moeten wij ook verantwoorden wat we gedaan hebben en wordt de user weer verwijderd. Dit komt misschien 3 keer per jaar voor in het kader van een release.

Use support zou ik tussen zowel de klant als de SaaS leverancier zetten. Dat is logischer.

Request fulfillment. Je zou moeten voorkomen dat dit soort service requests nodig zijn, dat zou weggeautomatiseerd moeten worden maar soms moeten er dingen geregeld worden. Ik zou het ook zo doen. Wijzelf zijn nog niet zover, maar als ik naar Salesforce kijk is er bij Salesforce een loket voor. Je kunt een contract afsluiten dat je dat loket mag bevragen.

Problem management. Ik zou niet weten waarom je deze zou schrappen. Ik vind dat binnen de SaaS-provider problem management geïmplementeerd moet zijn. Het is nog steeds zo dat er van alles gebeurt in gebruik waarbij je moet distilleren wat repeterend is en hoe dat opgelost wordt. Dat lijkt mij meer basis. Problem management gaat uit van een repeterende reeks incidenten die gedetecteerd worden en je er dan structureel aan gaat repareren.

In het framework mis ik geen processen, behalve wat ik net schetste van Salesforce: die successieplanners en meer het aanbodgedreven werken. Het feit dat de SaaS iets is wat voor een grote groep gebruikers is en aanbodgedreven gemaakt wordt en dan het consumeren van het nieuwe aanbod zit er niet in. Dat is wel iets wat steeds belangrijker wordt.

APPENDIX K: VALIDATION REVIEW COMPANY D

To perform the framework validation case study, an interview has taken place. The introduction text of the case study refers to external sources which have additionally been used to prepare the case study and verify some of the subjects that were mentioned in the interviews.

Interview		
Code	**Position**	**Date**
D-I1	Software Engineer & Architect	19-11-2012
D-I2	Enterprise Content Management Architect	22-11-2012
D-I3	Service Coordinator Mendix	22-11-2012

Table 8: Interviews held in the validation within company D

Documentation		
Code	**Title**	**Date**
D-D1	Process description Incident management	04-07-2011
D-D2	Process description Change management	22-03-2012
D-D3	Process description Configuration management	23-03-2012
D-D4	Process description Release management	12-10-2011
D-D5	Presentation “AO Service Offering in the Cloud Era”	11-2011

Table 9: Documentation used in the validation within company D

The interview reports are not literal transcriptions of the interviews, but mentioned aspects are partly summarized and shuffled to present a more logical story. As much information as possible is included in the interview reports to not accidentally overlook important aspects that may have gotten lost when summarizing. The digital recordings of the interviews as well as the mentioned documentation are available for inspection with the author.

INTERVIEW REPORT D-I1

Interviewee position: Software Engineer & Architect
Interview date: 19-11-2012
Interview duration: 72 minutes

Introduction
Ik werk nu ongeveer 16 jaar bij Cap, met alle rechtsvoorgangers/overnames. 3 jaar geleden overgegaan van KPN Getronics naar Capgemini. Meestal heb ik in maatwerkpakketten gezeten van Oracle; Oracle Databases, Oracle Forms, Oracle Designer. De hoofdklanten waren Makro en Metro wereldwijd. Voor die klanten heb ik de applicaties onderhouden en nieuwe implementaties gedaan. Sinds ongeveer 4 à 5 jaar hebben we als Capgemini een eigen pakket: RM3. Dit beheren we als SaaSoplossing binnen Cap, maar het staat ook on-premise bij klanten. Dit is ontwikkeld als on-premise applicatie en wordt nu hosted aangeboden als SaaS.
Sinds een halfjaar ben ik bezig met Mendix beheer, onder andere EBN, PostNL en NS HiSpeed. Het EBN zit nu in de transitie naar ons toe.
Ik doe heel veel dingen, de ene keer noemen we het Software Engineer, dan weer als Software Architect of Consultant.

Onze klant EBN heeft de hardware voor Mendix ge-outsourced naar Nobel en wij doen de Mendixapplicaties.

SaaS management framework
Op strategisch niveau bepaal je of je het model-driven gaat bouwen of standaard. Als je verwacht dat je applicatie snelle veranderingen door moet lopen of een wegwerpapplicatie wordt ga je eerder naar een model-driven applicatie toe dan wanneer je al 10 jaar vooruit kunt kijken, want dan ga je iets heel solide bouwen met misschien de verwachting elk jaar een release te doen.
Het proces waarin je deze keuze maakt staat boven de processen waarin het uitgevoerd wordt en daarom heb je op strategisch niveau geen verschil in traditionele en SaaS/model-driven processen.

Contract management speelt niet alleen tussen de klant en SaaS leverancier, maar ook tussen de SaaS en IaaS leverancier. Wij hebben nu bijvoorbeeld met postNL dat de klant een contract heeft met ons én met Mendix voor de cloudomgeving. Wij zijn geen tussenpersoon. Voor een aantal klanten functioneren we wel zo, maar als je heel specifiek kijkt is het niet zo. De SLA is afgesloten tussen Mendix en de klant. De klant gebruikt de Mendixcloud en wij voeren beheer uit op de applicatie. Dingen als server backup is de verantwoording van Mendix. Een softwarefout in de cloudomgeving is de verantwoordelijkheid van Mendix. Zelfs in beheer willen we er bijna niet tussen zitten, omdat wij er geen garantie over af kunnen geven. Je ziet vaak dat een klant een probleem bij ons meldt en wij dan met zijn SLA doorgaan naar Mendix. De klant ziet alleen dat zijn systeem niet opkomt en weet niet waar dat aan ligt. Wij schieten dan een ticket in bij Mendix. Mendix zit op PaaS en IaaS niveau, omdat zij hun hardware deels ook weer uitbesteden. Mendix heeft twee verschillende soorten cloudomgevingen: een standaardcloud en een “op maat”-cloud. Één van beiden is ergens anders gehost.
Als je Mendix als platform (PaaS) ziet en Capgemini meer als service (SaaS) klopt het framework wel.

Service catalogue management is van toepassing. In het geval van Mendix staat de hele omgeving bij Mendix op de machine en daar kun je ook de producten zien van de klanten waar jij mee te maken hebt. Capgemini bouwt ook generieke services die we in de markt zetten. Deze staan dan in de App store van Mendix. Dit kun je zien als catalogus. Mendix zelf, Capgemini en andere bedrijven hebben hier allemaal hun apps in zitten. Hiernaast is Capgemini bezig met een eigen pakket “MijnWoCo” voor woningcorporaties. Dit systeem gaan we als product verkopen in de markt. Je zou voor meerdere van dit soort producten een catalogus bij kunnen houden, het is dan alleen de vraag of iets anders in dezelfde catalogus komt als we iets anders erbij gaan ontwikkelen. Als je bijvoorbeeld iets voor een woningcorporatie en voor healthcare maakt is dat totaal verschillend. Als dit al samen in een catalogus komt zal dat de Mendix App store zijn.

Resource management op applicatieniveau doet de IaaS provider niet veel aan. Wij als beheerders controleren dat zelf. Wij doen de analyse dat er een tekort aan resources is op applicatieniveau en het toekennen doet dan vervolgens de IaaS. Als wij op dit moment niets doen crasht de boel en is het signaal: “je hebt wat nodig”. Resource management op IaaS niveau is meer het bepalen hoeveel applicaties/(virtuele) systemen ze op één groot fysiek systeem kunnen plaatsen en of daar uitbreiding voor nodig is. Op de totale hardware doen zij het dus en wij als beheerder doen het op applicatieniveau.
Dit zijn wel twee aparte processen, want het zijn verschillende dingen. Het is niet zo dat we dat op hetzelfde niveau doen. Deze twee processen communiceren wel met elkaar. Als een applicatie wil uitbreiden en meer geheugen nodig heeft dan er fysiek beschikbaar is moet de IaaS ook uitbreiden.

Planning and control zijn denk ik drie losse processen die iedere organisatie in het framework zelf uitvoert. Wij hebben een basisplanning voor onze standaard beheerdingen, maar komen er aanpassingen of incidenten die niet gepland zijn, dan is de klant de trigger voor onze planning. Dat is zowel vraag- als aanbodgedreven, want wij doen ook maandelijkse controles waarin wij punten kunnen zien die niet kloppen of iets in de backlog kunnen zien staan wat er al heel lang staat. Wij stellen de klant dan voor dat we dat gaan doen omdat het “dit en dit” verbetert. Het is een communicatiewerk tussen beiden, met name tussen de klant en service provider want die zijn heel nauw betrokken. Er kan ook zomaar van buitenaf bijvoorbeeld een regeringswijziging komen, zoals een veranderende BTW of ontslagvoorwaarden. De klant moet dan intern plannen hoe ze dat gaan doen en daarna komen ze naar ons toe en maken wij de planning op basis daarvan. Mendix heeft zijn eigen planning voor het platform. Dingen die wij graag gewijzigd zouden zien sturen we naar hun toe en dan is het voor ons afwachten in welke release het meegaat. We kunnen wel wat prioriteit geven maar ze hebben zelf een releaseplanning. Zij gaan de kant op van de vraag in de markt. Dat is nu bijvoorbeeld de opkomende mobile markt.
Ik denk dat de stippellijn tussen IaaS en SaaS groter is dan de stippellijn tussen klant en SaaS, maar het is niet zo dat de klant zegt: “volgende maand moet dit erin en jullie zorgen maar dat jullie de resources hebben”. Wij geven aan hoeveel werk iets kost en het werk zal dan ingepland worden. Het is een wisselwerking tussen klant en SaaS leverancier. In een traditionele omgeving stuurt de klant wat hij wil hebben en dan komt het met geluk over een half jaar of misschien over een jaar terug. Bij zo’n model-driven SaaS oplossing kunnen de iteraties maandelijks zijn.

Change management doen we in samenwerking met de klant. De input komt van de klant. Traditioneel kan het dan een half jaar of een jaar duren. Bij PostNL doen we dit maandelijks en krijgen we dus maandelijks een changelijst. We bepalen dan samen met de klant wat we in de volgende release mee kunnen nemen. De ontwikkeling van de change gaat dan vervolgens ook continu in samenwerking met de klant. Elke week doen

we aanpassingen en die nemen we gelijk door met de klant. Dus op het moment dat je daar al veranderingen in wil zien kan dat nog in dezelfde change aangepast worden. Er zijn wekelijkse iteraties. Ik vind dat change management overlap heeft met de klant in het framework maar niet met de IaaS want het platform en de onderliggende systemen hebben daar weinig mee te maken. Voor changes in het platform werkt dit op dezelfde manier; als wij een wijziging in het Mendix platform nodig hebben om de klant te kunnen bedienen dan geven wij een changeverzoek bij Mendix. Zij plannen hem dan in en wij kunnen wel wat druk uitvoeren, maar het is niet zo dat het direct gemaakt wordt tenzij de klant echt vastloopt. Dit loopt normaal in de reguliere releases mee. Wij hebben minder zeggenschap wat er bij Mendix gebeurt dan dat de klant bij ons heeft.

Change evaluation. We krijgen een change-omschrijving. Die beginnen we te bouwen en als het een grote change is laten we hem al aan de klant zien voordat we hem af hebben. De klant gaat dan valideren of hij voldoet aan hun wensen. Bijvoorbeeld een veldje dat rechts zit kan dan gewoon naar links geschoven worden, zonder dat het al hard gedocumenteerd is. Dit is meer de manier van agile werken als onderdeel van change management en er is geen los evaluation proces.
We doen wel maandelijks een validatie van het systeem en daarin kijken we of dingen aangepakt moeten worden, bijvoorbeeld voorstellen een proces voor opschoning te maken als wij zien dat de performance gaat dalen, of als één van de drie stappen in een proces handmatig is maar die eigenlijk beter geautomatiseerd zou kunnen zijn. We geven dan aan de klant aan dat we zijn werk makkelijker kunnen maken en daardoor krijgen wij weer werk om het te maken. Ik zou dit wel als een apart proces zien. Afhankelijk van hoe dynamisch de omgeving is gaan we regelmatig met de klant bespreken wat zowel wij als de klant aan wijzigingen zien. Het is dus meer een proces van bespreken. Je zou het proactief beheer of proactieve advisering kunnen noemen.

Traditioneel is meer dat je pas wat gaat doen als het fout loopt.

Knowledge management: wij hebben alle hoofdprocessen binnen de applicatie beschreven en degenen die erop werken moeten dat ook minimaal weten. Je moet zeg maar de flows en de samenhang weten, maar niet in detail welke statusovergangen of welke volgordes er zijn. Dat is wel handig, maar als je 30 applicaties in beheer hebt ga je dat niet allemaal onthouden. Het hoofdniveau documenteren we. Het detailniveau wordt niet gedocumenteerd omdat dan tijdens/na de ontwikkeling altijd de documentatie aangepast moet worden. Het verband tussen software en documentatie gaat dan verloren als je een aanpassing in de software vergeet aan te passen in de documentatie. Hoofdlijnen blijven altijd wel bestaan. Details hoeven ook niet gedocumenteerd te worden, omdat het voordeel van modelleren is dat je in het model al heel vaak de statusovergang ziet, je ziet wat er staat en als er uitzonderingen zijn dan hopen we dat de programmeur dat in een commentaarblok erbij heeft gezet.
Ik heb nu wel bij zóveel applicaties gezien dat de documentatie gewoon achterloopt. Vanuit de documentatie ga je er vanuit dat het zo werkt en als je daar dan in de programmatuur op voortborduurt kan het fout gaan als iemand toch een andere afslag heeft genomen. Het is ook echt de agile manier van werken om niet tot in detail te documenteren.

Configuration management: op applicatieniveau houden we wel versies bij. Wat in productie staat is gewoon één versie en daarnaast hebben we een source database die geleverd wordt door Mendix. Daar heb je alle releases in staan. Dat is meer release management. Dat is op applicatieniveau, niet op onderdelen van de applicatie. Je kunt niet binnen één applicatie meerdere verschillende releases naast elkaar hebben (2 modules binnen een applicatie met verschillende releases).
Het enige waar we configuration management eigenlijk voor missen op dit moment is als je modules uit de App store gebruikt bij het maken van een applicatie. Deze componenten kun je dan binnen je applicatie gebruiken (bijvoorbeeld e-mailing). Het probleem is dat je hier geen versienummer bij ziet, dus na verloop van tijd gaan ze het component upgraden naar een nieuwere versie maar kun je niet meer zien welke versie je gebruikt hebt. Het component wordt niet automatisch ook in je applicatie geüpdatet. Ze zijn hier nu wel mee bezig. In onze beheerchecklist hebben we nu gewoon de componenten staan en proberen we achter de release te komen die gebruikt is. Dat heeft er met name mee te maken dat als Mendix upgradet naar een nieuwe versie, een aantal componenten ook geüpgraded móeten worden. Als je de oorspronkelijke versie dan niet weet is dat lastig. Wij gaan configuration management in ieder geval gebruiken.

Software management: als je naar een multi-tenant service als Salesforce kijkt heb je geen distributie, maar Mendix is dat niet. Je kunt Mendix op verschillende plekken installeren. Voor MijnWoCo zijn we nu ook in

onderhandeling met Mendix en misschien gaat Mendix multi-tenancy ondersteunen en anders gaan we het in MijnWoCo zo aanpassen dat hij multi aankan (in latere versies). Op dit moment is het zo dat je een applicatie-instantie hebt die je voor een tweede klant op een tweede cloudomgeving opnieuw moet installeren. Het kan dan ook best zijn dat dit verschillende versies van de applicatie zijn. Dan krijg je wel te maken met software management. Dit is heel traditioneel. Dit heb je met name als je een standaardpakket gaat maken, niet bij een maatwerkapplicatie voor een specifieke klant. Standaardapplicaties heb ik nog niet gezien op Mendix, het enige waar dit naar voren komt is bij MijnWoCo.
Bij Mendix hebben we altijd minimaal een acceptatie- en een productieomgeving. Er is deployment tussen deze twee omgevingen. Iets kan vanuit acceptatie naar productie, maar nooit rechtstreeks naar productie.

De ontwikkelcyclus kan zowel iteratief als waterval zijn. Bij Mendix worden beide methoden gebruikt. Traditioneel is design groter en hoe wij het nu doen is realization en testing groter. Het is eigenlijk design en realisatie in één. Dit is meer de agile aanpak.

Use support hebben wij zowel bij de klant als bij de SaaS provider staan. Je hebt intern support bij de klant. De eindgebruikers nemen eerst contact op met hun eigen support desk, die hebben meestal functionele beheerders (bijv. bij PostNL) die gaan kijken of zij het aan kunnen passen met de functioneel beheeropties die ze hebben. Komen zij er niet uit, dan wordt hij doorgezet naar ons. Wij hebben eigenlijk met niet één eindgebruiker direct contact; niet in het geval van incidenten enzo. Wij hebben dan contact met de functioneel beheerders of key-users.

Request fulfillment is geen overbodig proces, want je krijgt altijd wel vragen om extra informatie te leveren ofzo wat niet onder incidenten valt. Bijvoorbeeld: “hoe werkt dit proces”. Dit is geen probleem of incident; er gebeurt niets. Het is meer iets wat ze bijvoorbeeld nodig hebben om later in change management een change op te voeren voor de eindgebruiker. De klant is dan bezig een wijziging aan te vragen. Hoe het dan precies werkt zou een vraag kunnen zijn; wat je normaal gesproken uit documentatie kunt halen zou dan een request kunnen zijn.

Access management: in principe doen wij niets met gebruikers toevoegen. Dit doen de functioneel beheerders. Zij assigneren gebruikers aan een rol. De rollen zijn dan wel weer gemodelleerd door ons. De eerste user, de beheerder, maken wij aan. Ik denk dus wel dat dit proces hier moet blijven staan.

Problem management valt bij mij meestal binnen incident management. Ik heb weinig ervaring met problem management als apart proces. Het is niet mijn ervaring dat als een ding vaker voorkomt of een incident waarvoor ze niet snel genoeg een oplossing zien terwijl er wel een workaround is wordt doorgeschoven naar problem management. Wel dat het verschuift naar change management en het binnen dat proces opgepakt wordt. De analyse gebeurt wanneer een incident binnenkomt en als ze er niet uitkomen of het duurt te lang zie ik hem vaak doorgaan naar change management. Daar vindt dan de verdere analyse plaats.
Problem management is een minder bekend proces.
Ik kan me wel voorstellen dat problem management een issue oppakt dat vaker voorkomt, maar vaag is.

Monitoring and diagnostics kan inderdaad de IaaS en de PaaS doen. ICT operations management doen wij ook. IT Operations control: jobs zijn ingesteld door ons, door de applicatie. Mendix zorgt standaard voor backups. Er zit een fail-over structuur in die als het goed is op hardwareniveau is geregeld.
Event management staat naar mijn verwachting op de juiste plaats bij zowel IaaS als PaaS.

Accounting and billing gaat vanuit de IaaS naar Mendix en door naar de klant. Ook vanuit Capgemini gaat het door naar de klant. Dit zit dus niet alleen bij de hardware-leverancier.

Service lifecycle management is in ieder geval van belang bij de klant en de SaaS provider (80%). Bij de IaaS (20%) zullen ze ook rekening moeten houden met de wijzigingen in de service als de service daardoor bijvoorbeeld meer of minder capaciteit nodig heeft. Het proces zit dus over de hele breedte en staat op de goede plaats.

Het enige wat ik mis in het framework is dat dingen zowel geïnitieerd worden vanuit de klant als vanuit SaaS leverancier voor het change management gedeelte. Het is zowel vraag- als aanbodgestuurd.

Change Management
Eerste impactanalyse en prioritering wordt door de klant gedaan. Binnen beheer is planning van belang.
De urgente procedures houden bij ons in dat in principe alles wordt stilgelegd en aan de prio 1 of prio 1 met blocking issue wordt gewerkt. Het probleem wordt in zo'n geval zo snel mogelijk opgelost. Hoe de urgente procedure werkt hebben we niet hard op papier staan, maar ik kan me voorstellen dat dat iets anders werkt dan de normale procedure. Toevallig hebben we er afgelopen vrijdag nog één met spoed in productie moeten zetten, omdat er iets gedaan moest worden wat niet meer kon.

De ontwikkelvolgorde is normaalgesproken als volgt: je modelleert een onderdeel, die ga je testen, dan doe je nog 2 à 3 kleine wijzigingen, die doe je in een iteratie, dan ga je deployen, die deployment gaat de klant testen (die loopt het hele traject door) en aan het einde krijg je de volledige deployment.
Wij doen minimaal 3 deployments na acceptatie voordat hij in productie gaat, dus het is heel iteratief.

De connecties naar software management voor de release en configuration management/de gebruiker voor de wijzigingen kloppen. De lijn terug van "does it work?" naar de impact analyse gebruiken wij niet. Een lijn terug naar modelleren is voldoende, maar het kan dan wel zijn dat je iets anders gaat bouwen dan oorspronkelijk in de impactanalyse was. De impactanalyse/documentatie wordt dan niet meer aangepast.

Configuration Management
We hebben een configuration plan, het datamodel wordt daarbij gebruikt. Naming conventions zijn meer vanuit Mendix opgegeven. Binnen een applicatie proberen we dit sowieso aan te houden. Mendix heeft een advisering hierover.
Configuration management op Mendix is minimaal. Het enige wat je kunt doen is een aantal constanten in de cloud updaten, daar is een lijstje van (5 tot 10 items). De documentatie staat dan in de cloud. Zo'n constante is bijvoorbeeld de URL naar een webservice of over hoeveel dagen data verwijderd moet worden. Dat soort configuraties kun je per instance aanpassen.

Statusmonitoring en verificatie doen we niet binnen een configuration management proces. Het hele versiebeheer wordt afgehandeld door het Mendix platform. Dit is dus geautomatiseerd.
Configuration management staat op de cloudomgeving. Wij hebben geen aparte omgeving waar we dit bijhouden.

INTERVIEW REPORT D-I2

Interviewee position: Enterprise Content Management Architect
Interview date: 22-11-2012
Interview duration: 55 minutes

Introduction
Ik ben heel veel in de operatie bezig geweest op het gebied van alle zijden van beheer. Zowel procesmatig als op architectniveau. Ik heb er wel kijk op, ik heb ook alle ITIL certificaten en dat soort dingen. Ik heb destijds met Jasper gekeken hoe we dat nou gaan doen met de cloud, want dat wordt een heel ander governance model. Je hebt in één keer allerlei andere leveranciers, je wordt echt een multi-vendor omgeving. Wij hebben gekeken of dat überhaupt impact heeft op ons model. We hebben daar toen heel erg over nagedacht, intussen is het een beetje weggezakt dus als we erover gaan praten komt het waarschijnlijk wel weer naar boven.
In de presentatie "AO Service Offering in the Cloud Era" ging het meer over dat cloud vendors een hele grote bedreiging voor ons zouden kunnen worden, omdat het applicatiebeheer dat wij doen steeds meer bij de cloud vendor terecht zou komen. Je hebt het dan meer over SaaS en PaaS oplossingen. Onze constatering was dat ons antwoord daarop zou moeten zijn: SaaS-oplossingen zijn over het algemeen gewoon point-solutions, individuele oplossingen, maar als je kijkt naar de behoefte van de klant is dat vaak vrij uniek. Aan die behoefte wordt voldaan door het op een unieke manier aan elkaar knopen van de verschillende point-solutions. Het idee is dat je dan nog steeds een system integrator nodig hebt die juist al die point-solutions aan elkaar knoopt en daar het geheel over bewaakt. Bij wie moet de klant anders aankloppen straks? Stel dat je productieketen is belegd over een heleboel verschillende applicaties bij verschillende vendors, dan gaan klanten steeds meer behoefte hebben aan één partij die het geheel monitort. Daar wordt een geheel nieuwe soort dienstverlening nodig.

Mijn functie momenteel is Enterprise Content Management (ECM) Architect.

SaaS management framework
Ik zie ASL (strategisch) terugkomen in het framework. Met het WoW framework ben ik bekend.
Ik zie Capgemini als een System Integrator en niet als een SaaS of PaaS vendor. Ik denk het volgende: je hebt een klant en een heleboel leveranciers. Een deel van die leveranciers is on-premise/pakket leverancier (Oracle, OpenText, etc.) en de anderen zijn cloud leveranciers (Force.com, Amazon, etc.). Een klant wil niet met al die verschillende leveranciers gaan praten en zoekt dus een system integrator met wie hij kan praten en die vervolgens met al die anderen kan praten. Als je kijkt naar het uiteindelijke applicatielandschap van zo'n klant zul je waarschijnlijk zien dat je een deel on-premise hebt en een deel in de cloud. De klant gebruikt al die applicaties, maar er zitten misschien ook nog relaties tussen die applicaties. Sommige zijn gewoon stand-alone. Ieder van die applicaties wordt geleverd. Sommige zijn maatwerk en sommige zijn pakketten. Ik zie de system integrator vooral als degene die het overzicht eroverheen houdt. De connecties tussen pakketten vind ik vooral een rol van de system integrator; die zorgt ervoor dat connecties geborgd blijven en voor de link met vendors. Bij een cloud vendor als Amazon hoef je echt niet te komen met: "ik heb nu een acuut probleem". Zij hebben pas echt een probleem als hun datacenter eruit ligt, maar dit zijn geïndustrialiseerde organisaties. De system integrator kan juist de balans vinden tussen klantgerichtheid (op maat, persoonlijk) en een stuk geïndustrialiseerd en innovatie. Ik denk altijd aan het model van i-cubed: intimacy, industrialization en innovation. Juist de system integrator zit bij de klant aan de intimacy kant, dus die weet precies wat er voor de klant speelt, hoe het landschap eruit ziet, waar het maatwerk zit en ook hoe de klantprocessen in elkaar zitten en hoe die dus gebruikmaakt van het landschap en als er iets uitvalt wat dat dan betekent voor die klant. Je hebt dan nog een stuk industrialization en innovation. Innovation is meer het service portfolio. Zij weten precies wat voor cloudvendors er allemaal in zitten en zij hebben precies de juiste mensen in huis. Amazon.com is highly industrialized. Zij pakken eigenlijk de hele IaaS stack, zeg maar alles wat eronder zit. Wat je erop zet is helemaal aan jou. Force.com doet dat nog een stukje hoger. Salesforce is helemaal tot bovenaan. Wat je dus op een gegeven moment krijgt is: voor ieder van deze applicaties krijg je 3 lagen: infrastructuur, applicaties/platform en functioneel. Een deel hiervan zal bij de cloud vendor liggen en een deel zal bij de system integrator liggen. Als het bijvoorbeeld PaaS is (Force.com) en daarboven heb je een stuk maatwerk gebouwd, ligt het beheer daarop bij de system integrator. Functioneel beheer denk ik dat je altijd ook nog een stuk bij de klant hebt. Als je kijkt wat de system integrator van de toekomst volgens mij gaat doen dan is dat industrialization, innovation en intimacy services leveren en dat doet hij voor het hele landschap, maar dat doet hij in eerste instantie op applicatieniveau en afhankelijk van het type applicatie kan hij dus verschillende layers van dienstverlening pakken. Voor een IaaS (Amazon) wordt alleen het onderste gedeelte door de cloudleverancier voor zijn rekening genomen en de platform/applicatielaag doet dan bijvoorbeeld de system integrator, want daar is verder niemand voor. Het bovenste gedeelte wordt door de klant gedaan.
Voor een andere (bijvoorbeeld Salesforce) heb je misschien dat helemaal tot het bovenste niveau door de cloudleverancier gedaan wordt. Die verhouding is zeg maar verkennend per applicatie. Al het maatwerk komt sowieso bij de system integrator terecht. Daarnaast vind ik dat een system integrator als je naar de toekomst gaat kijken waarin landschappen steeds meer geïntegreerd worden en echt volwaardige landschappen worden in plaats van constellaties van individuele standalone applicaties, vooral de interfaces ertussen een heel nieuw type dienstverlening worden. Je moet zorgen dat dat landschap als geheel samenwerkt. Volgens mij is dat niet helemaal nieuw en doen we dat al tientallen jaren, maar het wordt steeds meer connected.

Ik zou niet system integrator op niveau van de SaaS/PaaS provider definiëren, maar welk deel de system integrator pakt is helemaal afhankelijk van wat voor soort applicatie je hebt. Stel dat je bijvoorbeeld naar Salesforce kijkt valt een heel groot stuk bij Salesforce. Het framework gaat over stacks (Amazon, Force.com applicatie, etc.) heen. Je hebt binnen zo'n applicatie bepaalde dienstverlening en je hebt ook over de applicaties heen een bepaalde dienstverlening.
Beheer gaat van oudsher (bijvoorbeeld als je kijkt naar service desks in ITIL) over het beheer van een heel landschap en niet individuele applicaties. Ik denk dat je op verschillende niveaus naar je beheer moet gaan kijken. Je hebt inderdaad cycles als impact analysis – design – realization, dat doe je in wezen grotendeels binnen één applicatie-stack. Tenzij je natuurlijk een hele keten hebt, dan moet je het over applicatie-stacks heen gaan realiseren. De vraag is dan wat er nieuw is aan de cloud. Waarom zou je voor een cloud stack/applicatie een ander soort diensten nodig hebben voor wat betreft beheer? Mijn conclusie was dat er eigenlijk heel weinig verschillen zijn; het is oude wijn in nieuwe zakken. Eigenlijk is cloud gewoon IT in de breedste zin van het woord. Het enige verschil is dat het kanaal waarop je het aanbiedt via internet is. Misschien zitten in de applicatie wat architectonische keuzes, zaken zoals multi-tenancy, maar het verschil

tussen echte “born in the cloud”-applicaties en andere applicaties is net zo groot als een ECM pakket en een ERP pakket. Het beheermodel dat je ontwikkeld hebt voor IT is nog net zo van toepassing. De processen waar de klant niets mee te maken heeft in een cloudomgeving hoeft hij met on-premise ook niets mee te maken te hebben. Eigenlijk dezelfde activiteiten die je op dit moment bij volledige on-premise applicaties of applicatielandschappen ziet zijn ook bij SaaS-applicaties aan bod. Mijn gevoel is alleen dat het bij verschillende partijen gebeurt. Dingen die je normaalgesproken op infrastructuurniveau ziet vinden nu plaats bij de cloud vendor, waar ze voorheen misschien bij de interne IT departement plaatsvonden. Misschien is de technologie die ze gebruiken ook iets anders, maar de activiteiten die ze doen moeten toch nog steeds gebeuren en ik heb tot nu toe nog niet echt nieuwe activiteiten gezien. Misschien dat je ze iets anders moet inrichten, misschien omdat het een andere technologie is, maar dat is eigenlijk toch bijna business as usual. De technologie staat al 30-40 jaar niet stil. Ik zag wel inderdaad dat er voor ons als system integrator waarschijnlijk een accentverschil gaat plaatsvinden, omdat je voorheen de situatie had dat alles bij de klant draaide en de system integrator het technisch applicatiebeheer/infrastructuurbeheer voor zijn rekening nam. Dit zou ook een IT afdeling binnen bij de klant kunnen zijn. Je had externe leveranciers van producten. Alle activiteiten om het geheel in stand te houden en te vernieuwen zijn nog steeds nodig, alleen nu je met meer cloud vendors te maken hebt zul je zien dat een hele hoop activiteiten naar de cloud vendor verschuiven. Bijvoorbeeld het applicatiebeheer dat voorheen bij een system integrator/inhouse IT afdeling lag ligt nu bij de cloud vendor. Maar doen zij dan iets anders dan dat wij toen deden? Nee, eigenlijk niet.
Wat ik bedoel is dat de activiteiten die je noemt volgens mij nog dezelfde activiteiten zijn als we kennen van ASL en ITIL. Het enige wat verschilt per applicatie is waar de grens tussen verschillende partijen ligt. Bij een volledige on-premise applicatie die gehost is bij Capgemini ligt dat bij Capgemini, het applicatiebeheer misschien bij Capgemini en het functioneel beheer bij de klant. Stel dat je nu een Salesforce hebt ligt het bij Salesforce en de klant en hebben wij als system integrator eigenlijk op dat niveau weinig meer te doen. De snelheid waarmee je dingen neer kan zetten en applicaties kan ontwikkelen wordt steeds sneller, dus het accent van de dienstverlening wordt verlegd. De oude ITIL en ASL modellen zijn wel gericht op het beheer van een volledig landschap en niet voor individuele applicaties, met een algemene servicedesk die alles overziet en verschillende solution teams, etc.
Volgens mij ga je zien dat de system integrator het on-premise-spul houdt en eventueel wat maatwerk bovenop PaaS-spul, enz. Dat had je vroeger ook al, alleen het applicatielandschap gaat volgens mij toenemen omdat je steeds makkelijker dingen bijschaalt in de cloud (pay-per-use). Je zult meer een system integrator nodig hebben die service catalogue management doet. Dat had je vroeger ook nodig, maar het wordt steeds belangrijker. Die klant heeft echt geen idee wat er allemaal te halen is en of dat allemaal aansluit, het is een enorme menukaart. De system integrator zal moeten helpen om de juiste cloud services te provisionen en die af te schermen.
Wat ik denk is dat je binnen de stack eigenlijk dezelfde activiteiten houdt; ik zie geen andere activiteiten dan die je op dit moment voor de huidige on-premise-stack zou hebben, alleen zit zoiets nu dus bij verschillende partijen. Ik denk wel dat je een overkoepelend proces tussen de stacks hebt. Daar vallen dan dingen in zoals bijvoorbeeld ketenmanagement; dat je goed weet dat als je bijvoorbeeld een business process hebt en dat proces loopt dwars over applicaties heen en als er één applicatie uitvalt stopt je business proces, dan heb je een partij nodig die weet hoe die keten in elkaar zit voor dat business process en wie er dus gebeld moet worden. Je hebt iemand nodig die in eerste instantie het hele landschap kan monitoren en die ook precies weet hoe de business gebruikmaakt van dat landschap en dus kan ingrijpen. Voor het beheer is het dus zorgen dat de continuïteit geborgd is om op het moment van problemen adequaat in te kunnen grijpen. Dat is niet nieuw, want als je kijkt naar Belastingdienst, UWV, SUWI-keten, dan heb je allerlei verschillende organisaties. Dat heet ketenbeheer, dat is in de IT al 30 jaar een bekend begrip. Dat is allemaal al uitgedacht, alleen het gaat steeds belangrijker worden omdat je dus steeds makkelijker ketens kunt samenstellen met cloud vendors. Wat daarnaast denk ik steeds belangrijker wordt en wel nieuw is is dat je een soort consultancy krijgt dat je weet welke providers je kent en wat je eraan hebt en welke je snel even in je applicatielandschap kunt opnemen. Je moet in wezen vrij flexibel applicaties kunnen toevoegen, want dat is natuurlijk het grote voordeel van cloud; dat je kunt zeggen: “stop morgen maar met die applicatie”. Dat is in de praktijk natuurlijk weerbarstiger omdat er allerlei data in zit en je hebt allerlei maatwerk. We zijn er absoluut nog niet.

De processen waar een vraagteken bij staat zijn absoluut nog nodig. Software management ligt dan bijvoorbeeld bij een cloudleverancier waar het voorheen bij ons lag voor een on-premise applicatie. De cloud vendor heeft natuurlijk een hele batterij aan servers en bepaalde updates van componenten moeten daar natuurlijk over verspreid worden. Niks nieuws; voorheen lag dat bij system integrators en nu bij cloudleveranciers. Er is een groot verschil tussen configuratiebeheer op infrastructuurniveau (daar gaat het om:

"waar staan mijn PC's en servers en welke disks zitten daarin", dat kan heel snel wijzigen) en configuratiebeheer op softwareniveau. Bij de laatste denk ik veel meer aan releases en stukken software (componenten) met versiebeheer. Dat zijn twee heel verschillende werelden. Bij configuratiebeheer op softwareniveau heb je nog helemaal geen deployment. Als je dan die versie van de applicatie op een bepaalde server zet dan verandert configuration management van de infrastructuur (op die server staat dat). Maar ook dat is niet nieuw of specifiek voor cloud.
Accounting and billing valt over de gehele breedte, over alle lagen. Ook je hele kostenstructuur is zoiets waar zo'n system integrator enorm bij kan helpen. Een van de belangrijkste ideeën van Capgemini Immediate was dat je de klant niet wilt belasten met alle verschillende kosten- en afrekenmodellen van al die leveranciers. Een van de grootste producten die in Capgemini Immediate zat was enerzijds een hele standaardcatalogus van verschillende cloud vendors, die door ons gescreend waren en goed bevonden. Anderzijds was het een systeem dat de afrekenmethodes van al die leveranciers kon samenvoegen tot één rekening voor de klant. Dat gaat overdwars over alle lagen heen als infrastructure (Amazon) en applicatie (Salesforce) en daar krijg je ook system integrator kosten bij want je het gaat niet alleen over cloud, maar ook over on-premise en het samenvoegen van dat alles.
Een hele hoop processen kun je breder trekken, problem management ook. Je hebt vaak geen idee waar het probleem zit (welk deel van de stack), dat kan ook op hardwareniveau zijn.

Ik heb lang over cloudbeheer nagedacht en mijn uiteindelijke conclusie was best desillusionerend: er veranderd niets. Er komen wat accentverschillen en wat activiteiten bij andere partijen te liggen, maar verder blijft het hetzelfde. Ik vond dat eigenlijk een hele mooie conclusie.
Het framework is op een hoger niveau dan on-premise of SaaS; heel BiSL, ASL en ITIL is wat dat betreft geabstraheerd boven wat voor oplossing je nou hebt. Het zou mij ook verbazen als die drie ineens niet meer van toepassing zouden zijn. Volgens mij valt het reuze mee dat de cloud zo ontzettend anders is, maar daar zullen de wijze heren het ook niet over eens zijn.

Change Management
Ik weet wel iets van Mendix, maar ik ben daar niet heel erg in thuis.

Evaluatie gebeurt (vaak) niet in de praktijk.

Als de leverancier zijn applicatie gaat aanpassen zal hij toch echt de source code moeten aanpassen, dus daar zitten een build- en een testfase. Een SaaS vendor is iemand die een standaardpakket aanbiedt over internet; je gooit er een kwartje in en je mag het gebruiken. Model-driven development is met kant-en-klare bouwsteentjes functionaliteit in elkaar klikken. Een workflow management systeem is vaak model-driven; standaard workflowstapjes klik je aan elkaar. Force.com ondersteunt ook modelgedreven ontwikkeling; daar kun je applicaties bouwen met standaardcomponenten. Je hebt applicaties waarin je in plaats van bouwen kunt modelleren, daar hoef je de individuele bouwblokjes niet meer te testen, maar als je een bouwblokje wilt aanpassen zul je echt wel moeten gaan programmeren en testen. Als je kijkt naar een SaaS of PaaS leverancier valt build en test dus niet weg; als Salesforce een stuk functionaliteit wil veranderen in een applicatie dan gaan ze waarschijnlijk de source code aanpassen. In de meeste gevallen is build – test – implement voor mijn gevoel ook gewoon van toepassing op PaaS en SaaS. Als je de applicatie aanpast pas je de broncode aan. Sommige applicaties zijn echter gebouwd om door niet-technici gebruikt te worden om functionaliteit in elkaar te klikken. Dan pas je geen source code aan, maar modelleer je en kun je een aantal stappen versneld doen of overslaan. Testen lijkt mij altijd handig en implement is vergelijkbaar met deploy. Het aanpassen van de source code valt dan weg.

Het grote verschil tussen model-driven development en traditioneel broncode aanpassen zit naar mijn idee in de manier waarop je het doet. Je gaat veel meer prototypen in plaats van dat je de waterval methode gebruikt en eerst een hele RfC gaat ontwerpen. De lijn van "does it work?" naar model kan best een aantal keren doorlopen worden; je bent veel interactiever met de klant, samen het ding aan het bouwen en de specs aan het verzamelen. Je hebt dan van tevoren ook veel minder een RfC nodig, misschien moet je die dan eigenlijk achteraf schrijven: "dit hebben we gedaan". De klantvraag is aan het begin dan onvoldoende gespecificeerd, waar je vroeger zo'n keiharde waterval had dat de requirements eerste helemaal uitgekristalliseerd zijn met functional design is het met model-driven veel interactiever. Je krijgt een vage requirement en samen ga je aan de slag. Misschien wil je dan aan het einde wel even beschrijven wat je precies gedaan hebt, maar sommige systemen die beschrijven zichzelf al.

Eigenlijk mapt het weer helemaal met de oude situatie. Persoonlijk vind ik dat de sterkste conclusie die je kunt maken, want iedereen is heel erg bang voor de cloud maar dan kun je zeggen: “eigenlijk verandert er niet zoveel”. Dat is toch fantastisch!

INTERVIEW REPORT D-I3

Interviewee position: Service Coordinator Mendix
Interview date: 22-11-2012
Interview duration: 78 minutes

Introduction
Ik ben in 2008 bij Getronics gaan werken en ben mee-verkocht naar Capgemini. Ik heb een tijdje gewerkt als service delivery manager op een aantal contracten. Na ongeveer een jaar kwam ik tot de conclusie dat dat een veel te commerciële rol was voor mij, ik zit meer aan de klant-kant dan op de contractkant. Ik ben toen service coördinator geworden. Dat is wat operationeler en het heeft als voordeel dat ik met een team mensen kan werken. Dit was in de SAP-hoek, want daar ben ik begonnen en daar ben ik eigenlijk altijd gedelopyed geweest; met name technische projecten en een paar functionele. Ongeveer in september ben ik begonnen als service coördinator voor Mendix, dus het is voor mij nog vrij nieuw. Wat ik er heel leuk aan vind is dat het een hele korte omlooptijd heeft, bij die grote pakketten is dat gewoon niet leuk. Dit is een kwestie van een sprint van een paar dagen of een nieuwe applicatie in drie weken, in tegenstelling tot negen maanden. Dat vind ik echt heel leuk.
Ik kan wel wat zeggen over de manier van beheer (ik zit meer aan de beheerkant dan de commerciële kant) en ik ben in een ver verleden betrokken geweest bij het bouwen van een beheermodel voor de applicatie Baan, maar dat kun je eigenlijk op alle grote pakketten laten slaan. Modelmatig denken is mij niet vreemd en ik zie wel het een en ander aan verschillen tussen wat ik tot nu toe gewend ben en wat ik tegenkom in Mendix.
Ik ben bekend met het WoW framework van Capgemini.

SaaS management framework
Dat je service lifecycle management in de continual service improvement fase zet vind ik wel sterk. Het is natuurlijk wel zo dat juist die continuous improvement een heel sterk punt is van modelgedreven SaaS-oplossingen zoals Mendix. Dit is meer de agile manier van werken. Ik snap de redenering dat het proces daar staat als coördinerend proces.

Contract management: wat mij opvalt en dat vind ik één van de belangrijke verschillen tussen wat ik gewend was en wat ik nu tegenkom (en daar moeten we iets voor verzinnen) is dat je in een traditionele omgeving in de tijd gezien een vrij duidelijk stappenplan hebt: je hebt een lead te pakken, dan ga je het één en ander demonstreren of aanbieden en dan krijg je mogelijkerwijs een proof of concept en dan ga je het contract afspreken waarna het project op gang komt en ergens in het project wordt een beheercontract opgetuigd en dan gaat dat kalmpjes aan een keer lopen. De hele agile manier van werken is zó snel dat het contract er nooit komt. Ik denk dat wij al 4 of 5 applicaties in beheer hebben of in beheer zouden moeten hebben, maar we hebben nog geen contract. Alles loopt dus, maar Capgemini moet daar administratief of qua governance iets anders voor verzinnen want als je geen contract hebt heb je geen nummer en als je geen nummer hebt heb je geen service desk, kun je je uren niet kwijt, kun je geen resources claimen, etc. De uren moeten dus worden geparkeerd en als er een echt contract is weer worden overgeboekt. De meldingen kunnen nog niet in de service desk, want ik heb nog geen nummer, dus dat moet ergens apart bij mij binnenkomen in een sheetje. Het is nog niet agile. Ontwikkelaars stormen natuurlijk met een noodvaart door een project heen (een project kan tussen de 2 weken en 3 maanden zijn) en dan is het klaar voor de overdracht. Alleen moet er dan dus nog een SLA gemaakt worden en voorwaarden en onderhandeld worden. Je moet dus zorgen dat de SDM’er voortaan al aanhaakt op het moment dat iemand bedenkt dat we een project gaan doen, anders ben je te laat. De klant vindt het geweldig als je heel agile werkt; het liefst elke maand een nieuwe release en bijna elke week een sprintje, maar ze zijn nog wel heel traditioneel in hun eigen procurement. Op het moment dat je het namelijk eens bent over wat de klant wil moet de hele wereld een stempel zetten en eroverheen plassen natuurlijk. Het is ook lastig om daar vorderingen in te maken, want bij onderhandelingen moet het misschien zes keer heen en weer. We zijn al bezig met een standaard contract template: geen maatwerk, gewoon een menu waarin je goud, zilver of brons kunt kiezen en dat is het. We hebben 3 parameters: gewenst service level,

de grootte van de applicatie in bouwuren (vertalen we in functiepunten) en de complexiteitsfactor. De klant kiest het service level en aan de calculatiekant stop ik alleen het aantal uren en de complexiteit erin en dan rollen daar getallen uit. Dat moet helpen. Mendix zelf biedt volgens mij ook gewoon 3 mogelijkheden: zilver, goud en platina.

It service continuity management speelt inderdaad bij zowel service & platform management als infrastructure & hardware management. Eigenlijk zijn het twee afzonderlijke processen, het is hetzelfde proces maar het domein is anders. Het is wel handig als beide domeinen met elkaar communiceren. Het zijn echt specifiek andere dingen en ook weer niet, het hangt er heel erg vanaf vanuit welk perspectief je het bekijkt. Vanuit klantperspectief is het hetzelfde want het interesseert de klant geen moer waar het misloopt, hij weet dat hij niet krijgt waar hij voor betaald heeft. Als je het vanuit de service of de infra bekijkt zijn het twee verschillende dingen. Ik voel mij ervoor verantwoordelijk dat ik aan mijn klant het geheel lever. Als dat niet werkt komt hij bij mij. Dan gaan we kijken en zijn wij in de eerste plaats verantwoordelijk voor het verlenen van de service en daar doen we alles aan. Het kan dan zo zijn dat we ertegenaan lopen dat het een bug in het platform is. We zijn dan nog steeds aansprakelijk en accountable voor de klant, maar dan hebben wij er eigenlijk geen invloed meer op. Dat wij een klant hebben die zijn ding niet kan doen, daar heeft Mendix helemaal geen weet van tot op het moment dat wij gaan roepen: “Hé jongens, er is bij jullie iets aan de hand”. Dat loopt als het goed is altijd via Capgemini. Mendix is PaaS, maar voor sommige klanten ook de IaaS provider. Daar hebben we Mendix in de cloud draaien op hun infra en er is ook een variant waarbij klanten zelf de infra regelen en het platform lokaal hebben draaien. Dan heb je een losse infra- en platformprovider.
De grens tussen service en platform is eigenlijk heel blurry.

Waar resource management zit is maar net hoe je ernaar kijkt. Voor mij zit resource management in de eerste plaats bij service en platform omdat dat de resources zijn waarmee ik werk. Alles daaronder, met name op infragebied, dat moet er gewoon zijn. Daar vindt uiteraard ook resource management plaats, maar ik vind wel degelijk óók aan de services kant. Resources zijn dan de combinatie van mens en skill. Waar wij veel mee werken is wat we hier een IKA noemen. Dit is voor bijvoorbeeld SAP een heel uitgebreide, die geeft het zelfs tot onder moduleniveau aan. Dit is een matrix van personen en SAP module-onderdelen; wat het expertise-level is van die persoon voor dat betreffende onderdeel. In SAP is dat heel uitgebreid omdat je daar heel veel verschillende dingen hebt. In Mendix zal dat wat minder zijn, maar ook daar zul je mensen hebben die van een bepaald gebied veel weten. In die zin gaat het iets verder dan alleen maar plannen wanneer pietje op vakantie is en jantje weer terug is. Je moet ook wel weten wie wat kan. Ik zou resource management apart bij service & platform management zetten, omdat dat rechtstreeks invloed kan hebben op de dienst die je op een bepaald moment wel of niet kunt verlenen. Soms moet je nee verkopen omdat je niet de goede resources beschikbaar hebt.

Wat planning and control betreft heeft iedere organisatie uiteraard zijn eigen cyclus. Daar zit wel een verband tussen.

Modelgedreven ontwikkelen is echt leuk. Ik zat pas in een bespreking aan de telefoon en we konden aan de telefoon een incident oplossen door een verandering door te voeren.

Een SaaS-oplossing heeft wel distributie. Het proces is er wel, maar de instances zijn er maar één of hoogstens 10 of zo, maar je hebt dat wel degelijk. Ook al is het er maar één, dan is per ongeluk de werkelijkheid zo dat het er maar één is, maar het hadden er ook meer kunnen zijn. Er is een duidelijk verschil tussen Mendix en Salesforce, want bij Mendix is het niet zo dat klanten gedwongen over moeten naar een nieuwe release. Bij Mendix hebben alle applicaties een eigen releaseversie, dus er zijn applicaties die nog draaien op oudere versies van het platform; je moet daar echt kiezen van “ik ga nu over”. Als je maar één omgeving hebt is er geen distributie, maar in Mendix wel. Ook als je een aparte acceptatie- of testomgeving hebt heb je geen distributie, de acceptatieomgeving loopt dan gewoon iets voor.

Configuration management is bijvoorbeeld bij Mendix wel degelijk van toepassing op de software. Je moet weten op welke platformversie een bepaalde applicatie draait om bijvoorbeeld te weten of een bepaalde import- of exportfunctionaliteit beschikbaar is en of dat wel of niet goed werkt. Er zijn een aantal van dat soort grappen die daar toch wel belangrijk in zijn. Versiebeheer op het platform is belangrijk, verder is het eigenlijk niet zo heel spannend. Bij de infra kan het echt heel spannend zijn, hoewel dat tegenwoordig toch ook allemaal

modulair is. Op fijnmaziger niveau dan componenten vindt in ieder geval geen configuration management plaats en ik vermoed dat het alleen op applicatieniveau gebeurt.

Knowledge management is uiteraard van toepassing. Het proces wordt in de transition fase meer onderhouden en in de operation fase meer gebruikt. Hetzelfde geldt voor configuration management, dat zit ook een klein beetje aan de operationele kant. Als er incidenten zijn in een SaaS-omgeving kan het zo zijn dat je een workaround moet toepassen om het incident op te lossen die gevolgen heeft voor het configuration management of in ieder geval het kennismanagement. Voor mij is knowledge management documentatie, de database of known errors and solutions (die bak waar iedereen altijd zo enthousiast aan begint en die dan niet wordt onderhouden). Dit wordt wel degelijk in de transition fase ook onderhouden. De documentatie is er in ieder geval voor de ontwikkelaars en ik zou zeggen in een Mendix-omgeving zit daar eigenlijk niet of nauwelijks eindgebruiker-documentatie in. Dat heeft ermee te maken dat het ontwikkelproces zó kortcyclisch is en zo dicht op de gebruiker. Ik kan me niet zo goed voorstellen dat de ontwikkelaar documentatie voor de gebruiker moet maken, het is andersom. Traditioneel wordt eerst een functioneel ontwerp gebouwd dat eindeloos bediscussieerd wordt en daarna krijg je een technisch ontwerp en op een gegeven moment krijg je als ontwikkelaar een lijstje met specs wat je moet gaan bouwen. Dan moet je uiteindelijk als ontwikkelaar beslissen hoe je het gaat bouwen, je bent zelf vrij hoe je het precies bouwt en dat moet je dan documenteren voor de eindgebruiker. Aan de Mendix-kant is dat andersom. Die lui zitten bij elkaar en dan zeggen ze: “daar wil ik dat hebben staan”. Het grappige is dat gewoon de gebruiker of de functioneel beheerder aan de klantkant bijvoorbeeld aan de telefoon de business rules zit uit te leggen als we een sprint zitten voor te bereiden, ik verbaas me daar mateloos over dat dat niet is vastgelegd maar dat is gewoon niet zo. De klant weet het precies en bij ons is het dan “kloperdeklop, klaar”. Daar zit geen gebruiksaanwijzing bij.
Er wordt wel wat vastgelegd in het model zelf, Mendix biedt daar in ieder geval mooie mogelijkheden voor, maar aan de klantkant vraag ik het me af. Dat vind ik een beetje griezelig want de ene persoon bij de klant wil misschien het ene en zijn collega zegt misschien over 3 maanden dat het helemaal niet zo moet. Voor mij is het ook nog de vraag hoe we daarmee omgaan.

Het onderscheid tussen change evaluation en change management kan ik me theoretisch wel voorstellen. Ik denk dat dit proces wel speelt, maar aan de klantkant zit. Ik zie hem dan als een soort change board-achtig iets, maar dan wel een change board dat helemaal bij de klant zit want die trajecten zijn zó agile dat wij helemaal niet in het change board terechtkomen. De evaluatie gebeurt dan voor de change. Er komt bij de klant een lijstje van veranderingen uit, dat wordt geprioriteerd en dat komt naar ons toe met het verzoek er eens een estimate op te geven.

Ik vind het loskoppelen van request fulfillment van incident management omdat requests anders een lage prioriteit zouden krijgen flauwekul. Voor mij is het één proces. Ik vind use support daar een heel mooie term voor die je er beslist in moet houden. Ik vind dat als je het om díe reden uit elkaar haalt, dan heb je iets niet goed gedaan in de inrichting van je use support proces want je kunt natuurlijk gewoon afspreken met een klant hoe belangrijk dat is en hoeveel prioriteit dat moet krijgen. Er zijn een heleboel verschillende trucs voor; je kunt bijvoorbeeld een bepaald gedeelte van je resources daarvoor reserveren. Vanuit het framework (conceptueel) zou ik deze twee processen niet uit elkaar trekken. In de praktijk is het in mijn geval samengevoegd want ik heb twee consultants rondlopen op dit moment en als ik dat uit elkaar moet gaan halen heb ik een probleem. Je kunt het wel uit elkaar halen, maar dan stel ik me echt een club van 25 man voor waarvan er een aantal dedicated verzoeknummers doen. Ik vind dat niet logisch en ik vind het ook niet wenselijk, maar dat is meer vanuit knowledge management oogpunt want ik wil heel graag allround consultants hebben. Ik wil dat mensen zoveel mogelijk overal aan geroken hebben en van dingen op de hoogte zijn, dat werkt gewoon beter. Mogelijk zou je vanuit de praktijk een reden kunnen hebben om dat praktisch gezien te splitsen, maar procesmatig zie ik daarin geen onderscheid.

Problem management: ik merk dat ik nog een beetje vastzit aan het ITIL-denken. Incident management is ad hoc het probleem wegnemen. Voor mij is problem management nog steeds dat op meer gestructureerde wijze benaderen. Ik denk dat problem management er wel is, maar een kleinere rol speelt dan bij de grotere omgevingen, juist vanwege die agility. Het gaat allemaal zo snel en je bent zó snel in een sprintje bezig om een aanpassing te doen dat je een incident oppakt en eigenlijk al direct in de gaten hebt hoe je dat moet aanpassen óf dat gaat niet werken en je moet er iets structureels voor verzinnen en dat komt dan in de volgende release óf dat kan niet en we moeten het proces aanpassen. Soms heeft het wat meer impact en dan moet je een volgende release of platform-release afwachten.

Access management zou ik nog verder naar de klant toetrekken. Dat er bij de service provider mensen zitten die de handjes zijn, dat kan, maar het proces zelf zit bij de klant en bepaalt de klant. Het uitvoeren doet de klant ook nog vaak zelf.

Event management zit tegen de monitoring aan. Bij IT operations control zit voor mij nog iets waar ik nog niet van weet wat ik ervan moet denken. Er is iets anders mee dan wat ik gewend ben vanuit de SAP/ERP wereld. Wat in zo'n SaaS-omgeving gebeurt zit vaak veel meer aan de gebruikerskant dan aan de batch-verwerkende kant. ERP systemen zijn natuurlijk toch nog wel redelijk batch-georiënteerd en SaaS is dat eigenlijk niet. Je kunt wel batches bouwen maar dat is eigenlijk alleen maar als je iets hebt van: "het is wel heel groot en ik heb het niet onmiddellijk nodig", voor de rest doe je alles gewoon on-line. Dat IT operations control zit dan eigenlijk niet meer zo aan de servicekant. Voor het grootste gedeelte zit het aan de platform- en hardwarekant en in de services is dat eigenlijk alleen maar als er voor een bepaalde verwerking controlelijsten moeten worden gedraaid of zo.
De diagnostics die ik tegenkom hebben eigenlijk alleen maar te maken met: hoeveel gebruikers zijn er on-line, hoeveel memory wordt er gebruikt en hoe performt de boel.
Accounting and billing: een stukje financial management heb je bij service & platform management natuurlijk ook nodig.
Data management is infra, maar data integriteit heeft wel degelijk ook met het platform zelf te maken óf je moet zeggen: "dat laat ik afhandelen door de database engine zelf". Dat wordt vaak onder IS gezien.

Wat ik zie in de praktijk is dat de consultants een andere rol hebben dan in de ERP wereld. In de ERP wereld heb je veel meer compartimentjes van specialiteiten. De één zit veel meer aan de voorkant in het proces en de ander veel meer aan de achterkant. In het Mendix-gebeuren in ieder geval heb je veel meer beachvolleybal zeg maar. Daar lopen ze allemaal én voor én achter en ze vervullen allemaal wisselende rollen. Ik vind dat persoonlijk heel leuk. Voor mij is dat een reden om me af te vragen of zo'n framework als dit nou wel echt een goede fit heeft op een SaaS-omgeving. In dit uur zijn er een aantal momenten geweest waar ik dacht: "vroeger was het zo en dan past het allemaal netjes in die vakjes, maar in ieder geval met Mendix zijn de grenzen wat vervaagd en de rollen wat veranderd". Er zijn toch wel een aantal dingen waarvan je je kunt afvragen: "is het nou dit of is het nou dat". Voor mijn gevoel wringt het een beetje. Voor mij zou de vraag belangrijk zijn waarom dit framework een goede fit heeft op een totaal veranderd stukje dienstverlening.

Change Management
Prioritering zou gelijk kunnen zijn aan de planning. Je hebt zonder meer urgente procedures.

Ik denk dat je met het proces een heel eind bent. Er zijn wel een aantal onderdelen zoals filteren, prioriteren en doorgeven van de wijzigingen aan de gebruiker gebeurt niet bij ons, dat zit eigenlijk helemaal aan de klantkant.

Evaluatie: op een gegeven moment ben je klaar in de bouw (ik vind het al heel veelzeggend dat dat eigenlijk in de acceptatietest-omgeving gebeurt) en dan is er een soort oplevermoment wat eigenlijk ook een soort evaluatie is. Je loopt dan de hele handel door en dan wordt er in overeenstemming besloten: "oke, dit is het" en dan gaan we op de knop drukken.

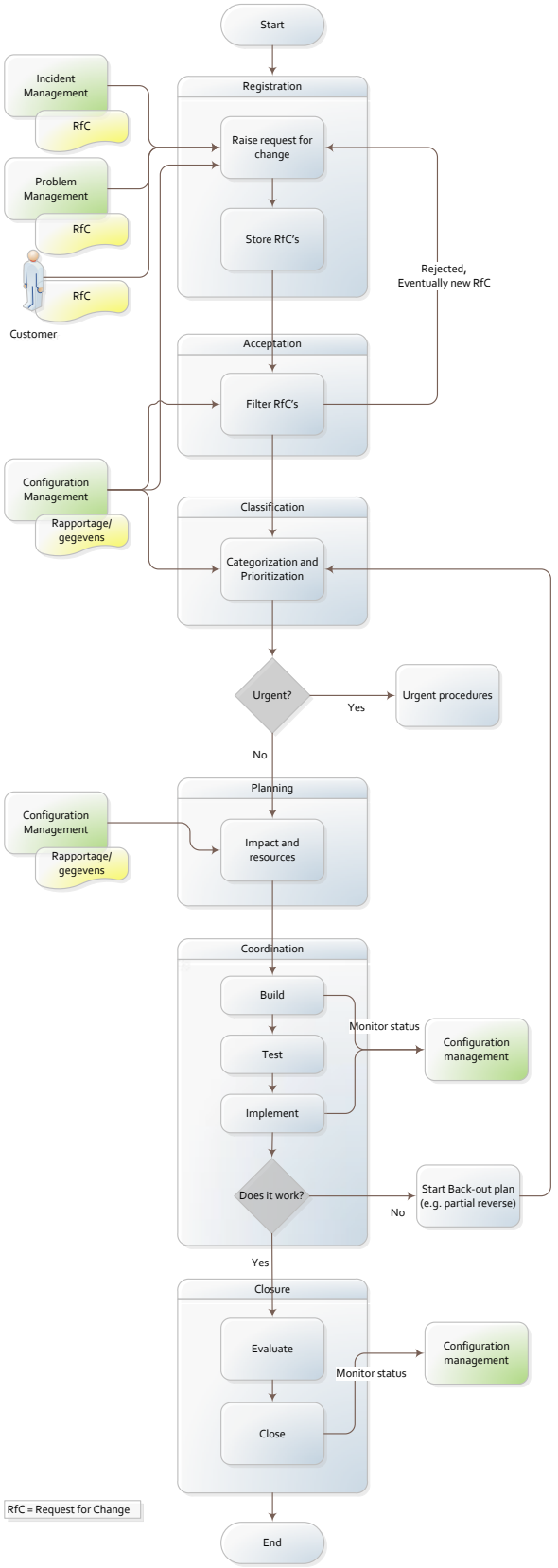
Wat betreft de connectie met configuration management wordt de nieuwe versie al veel eerder aangemaakt dan bij de close-activiteit. Op het moment dat je gaat bouwen maak je een nieuwe versie. Ik denk dat bij de model-activiteit ook een pijl naar configuration management zit. Als je een simpele omgeving hebt ga je je acceptatie-omgeving wijzigen bij de start van een sprint of bij de start van een nieuwe release. Die wordt dan uiteindelijk in zijn geheel opgepakt en over de productieomgeving heen gezet. Je hebt dus dan al een nieuwe release met de status "in ontwikkeling".

Configuration Management
Ik denk in ieder geval dat het niveau heel erg verschillend is. Configuration management gebeurt op een vrij hoog niveau. Dat is hoogstens het bijhouden van modules of aparte routines, maar verder niet.
Wat betreft het configuration plan kan ik me niet voorstellen dat wij veel verder komen dan: bij die klant met die applicatie gebruiken we versie dat en dat. Dat kun je een plan noemen, maar het stelt weinig voor.
Van statusmonitoring kan ik niet zeggen: "dat zit er beslist niet in", maar om het nou apart te definiëren in een procesmodel is een beetje over the top. Dat auditen gaat gebeuren kan ik me niet voorstellen, maar je weet het nooit. We willen bij Capgemini altijd alles netjes afgevinkt hebben.

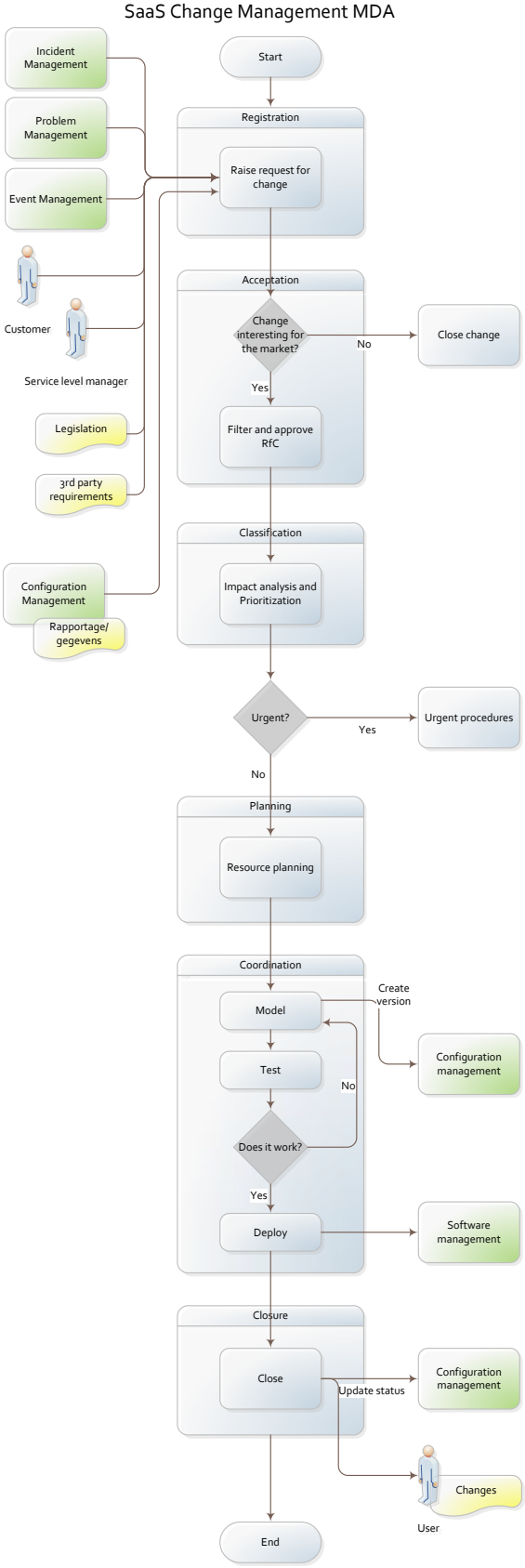
APPENDIX L: THE THIRD AND FINAL CONCEPT OF THE THEORY PROCESSES

THE CHANGE MANAGEMENT PROCESS

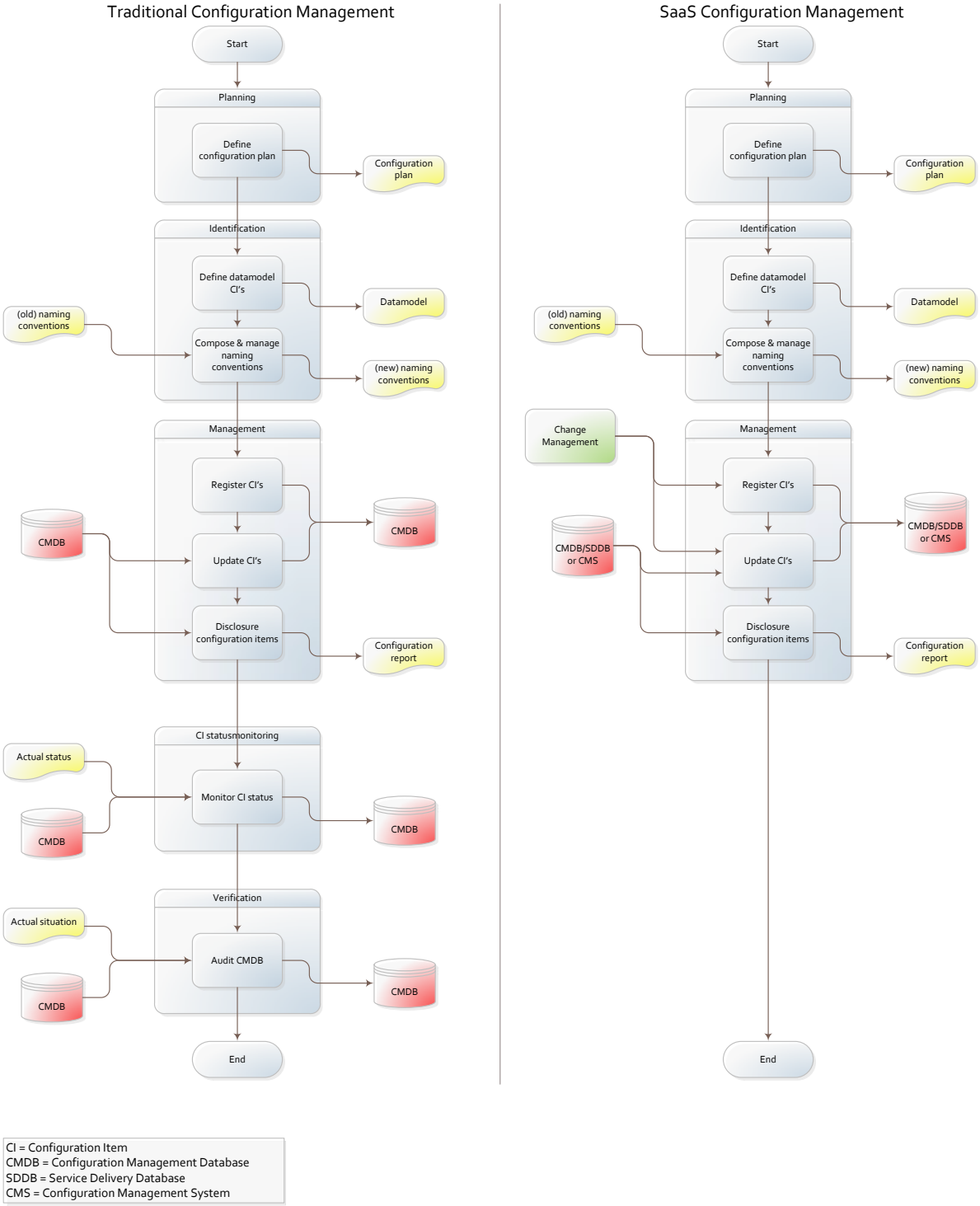
Traditional Change Management



SaaS Change Management MDA



THE CONFIGURATION MANAGEMENT PROCESS



CI = Configuration Item
CMDB = Configuration Management Database
SDDB = Service Delivery Database
CMS = Configuration Management System